1-1-2000

Mixed-signal functional level implementation of 1000BASE-X physical layer for gigabit Ethernet with synthesis of the PCS

Khaldoun A. Bataineh
Iowa State University

Mixed-signal functional level implementation of 1000BASE-X physical layer for gigabit Ethernet with synthesis of the PCS

by

Khaldoun A. Bataineh

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Major Professor: Marwan Hassoun

Iowa State University

Ames, Iowa

2000

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Khaldoun A. Bataineh

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ABSTRACT

Behavioral modeling, functional modeling and simulation for both digital and analog systems are powerful tools of verification in the mixed-signal (analog and digital) circuit design process. Recently approved IEEE standard 802.3z 1000BASE-X (Gigabit Ethernet) physical layer represents such a mixed signal system. The 1000BASE-X represents a complex and challenging mixed-signal system that transfers data at very high speeds. The system is composed of a Physical Coding Sublayer (PCS) that is entirely digital and a Physical Media Access (PMA) sublayer that is a mixed-signal subsystem but is predominantly analog. In this work, the entire 1000Base-X mixed-signal system was modeled. The digital parts using Verilog® and the analog parts using SpectreHDL® (Spectre Hardware Description Language). The top-level integration of the mixed-signal system was achieved using SpectreVerilog®. The entire physical layer was realized and verified in a 0.35μ CMOS technology. The PCS sublayer was synthesized down to the gate level using Synopsys®.

1. INTRODUCTION

1.1 Purpose of Chapter

In this chapter subjects covered by this thesis are introduced, those subjects include the importance of functional verification, mixed-signal simulation, Gigabit Ethernet and synthesis tools. Finally the thesis perspective is introduced at the end of the chapter.

1.2 Preface

1.2.1 Modeling and functional verification

In the design flow of any system, there is always a debate between two camps. There are those who are calling for spending enough resources on functional modeling and verification of the system before the start of the actual design process. On the other hand there are those who are calling to limit this step by deriving general specifications that experienced designer can determine their feasibility. The debate would go on to include different qualities that are found in one approach and not in the other. These qualities include probability of hitting a major problem later in the design process, time to market, optimization of the design... etc. We are not going to get into this argument of which way is better, but rather to show that it is possible to get the advantages of both.

Due to major advances in computer aided design (CAD) tools for digital systems, one only needs to add moderate design effort on the functional level

and then use different tools to get the final design. The tools for digital systems include synthesis, floor planning, placement and routing. All the mentioned tools are available on the market. A suggested design flow diagram for digital systems that describe the order of using the tools is shown in figure 1.1.

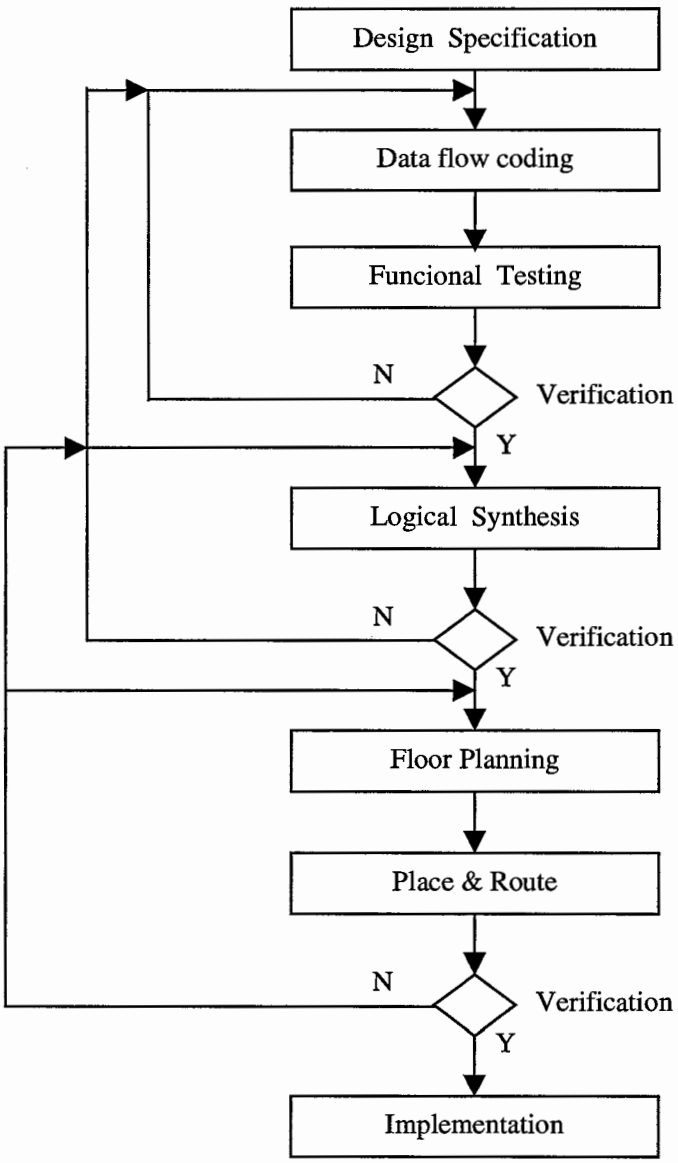


Figure 1.1 Digital design flow diagram.

For analog systems a design process is shown in figure 1.2. Most of the steps are custom done for a specific design. Although the number of elements in an analog design, in general, is smaller than those for a digital design, it is usually more time consuming. Analog designs, as will be mentioned later,

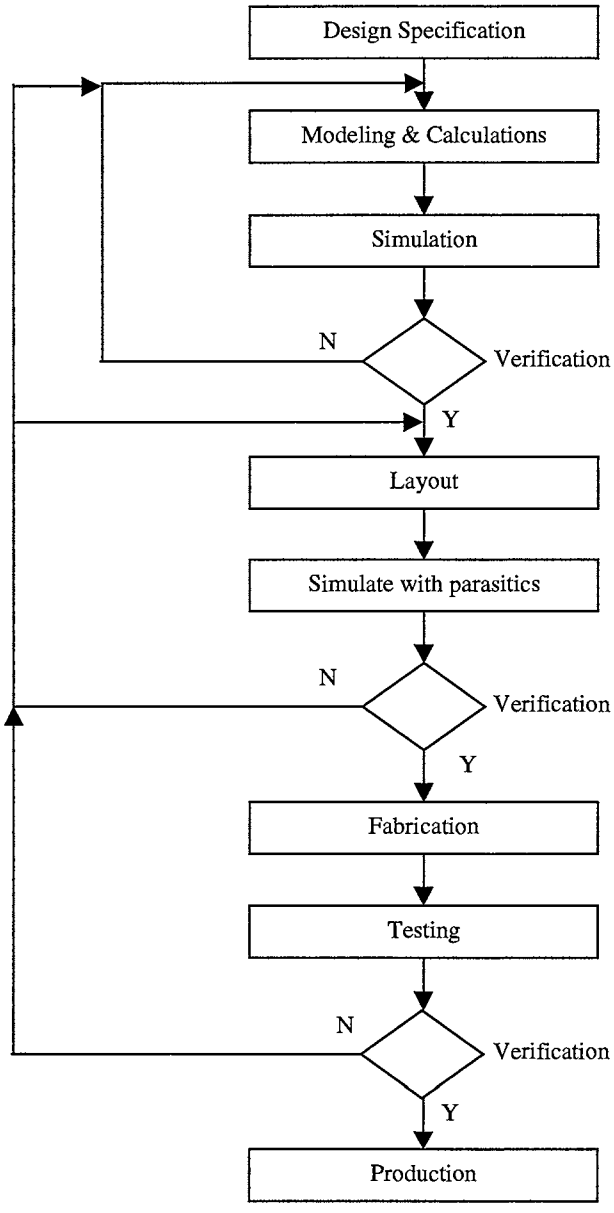


Figure 1.2 Design flow diagram for analog systems.

are more complex than digital designs, thus making them more difficult to synthesize. Despite the previous reason, there are signs and trends in the market in addition to research work that indicate that analog synthesis is only a matter of time [1,2].

1.2.2 Mixed-signal simulation

In reality all the circuits are the same analog or digital. The same laws of physics apply to both. Due to limitations in the analog simulators' speed and size handling capabilities, digital simulators use functional characteristics of the digital circuits in simulation instead of physical characteristics. Functional simulation is faster and can handle larger circuit sizes.

Mixed-signal systems are systems that have a combination of analog and digital circuits. Many of today's applications are classified as mixed-signal systems such as digital-to-analog converters, analog-to-digital converters, digital sound synthesizers, and data networking systems, to mention a few.

Using a digital or an analog simulator to run a simulation on a mixed-signal system is impractical, each of the simulators has its own inherit limitations and features. Mixed-signal simulators use a combination of both analog and digital simulators together. More about mixed-signal simulators is covered in section 1.2.

The Gigabit Ethernet 1000BASE-X is a data networking mixed-signal system. It has both analog and digital parts. To be able to simulate the system as a whole we need a simulator that can simulate mixed-signal circuits. More about the Gigabit Ethernet is covered later in sections 1.2.3 and 1.4.

1.2.3 Gigabit Ethernet

This is the latest generation of the Ethernet standards series by the Institute of Electrical and Electronics Engineers (IEEE). Ethernet, which is defined later in section 1.3, is a local area network wired communication technology [6]. Gigabit Ethernet has different varieties that are classified according to their media: 1000BASE-LX and 1000BASE-SX across fiber; 1000BASE-CX across Cat-3 copper; and 1000BASE-T across Cat-5 copper [3]. The justification for Gigabit Ethernet upgrading from the previous versions (working at 100 MHz) is set in the "Five Criteria" by its study group [3].

1.3 Mixed-Signal Simulation

1.3.1 Digital simulation

The main reasons for the invention of Hardware Description Languages (HDLs) are simulation, documentation, and recently synthesis [7]. Synthesis essentially is the process of transforming functionality, which is initially described in HDL, to an optimized technology-specific netlist [11]. Simulation gives the designer the power to verify the design, and enables the

testing of different modeling variations before realizing the design. Documentation enables the designer to re-use his/her or others parts of the design for future designs. Synthesis is achieved by adding a level of effort during coding, see figure 1.1 for the design steps of digital systems, more about the levels of abstraction in section 1.5 about the synthesis.

HDLs have a different nature than the normal programming languages such as FORTRAN, Pascal and C that are sequential in nature. It has a parallel nature, which enables modeling of concurrency that describes the hardware elements [5].

Verilog® HDL is a general-purpose hardware description language. It has the ability to simulate large digital designs with adequate speed compared to other HDLs such as VHDL. Verilog® HDL supports hierarchical building of a design which is a natural bottom-up flow of design. Circuits can be described at many levels, figure 1.3 shows many of those levels classified according to the abstraction level [5], where the abstraction level describes the system elements interaction details.

Verilog® HDL supports many levels of abstractions, from system level down to switch level [8]. Finally this HDL is supported by many popular synthesis tools such as Synopsys®, which supports the register transfer level (RTL) description, it is also called the dataflow level, where the description of how data flows between registers and how data is processed between

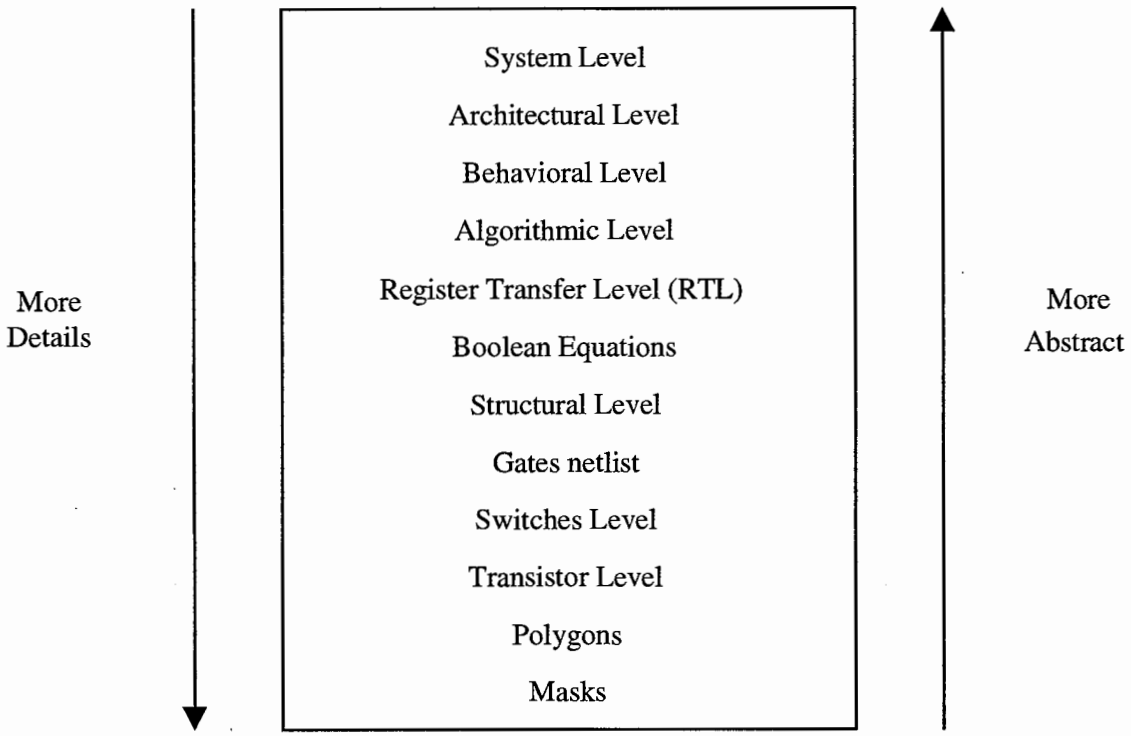


Figure 1.3 Some of the levels of circuit abstraction [7].

registers. In the behavioral level of abstraction, only the desired design algorithm is defined, no concern about the hardware implementation.

Digital simulators use logical functionality of the circuit in simulation rather than the physical laws to achieve higher speed and more size handling capacity than analog simulators for the following reasons. The logical functional description has two discrete states (true and false) with one variable for each node compared to continuous states for analog (continuous signal) with two variables (voltage and current). In digital circuits usually signal flows in one direction throughout the circuit compared to undefined

signal flow direction through analog circuits that even can change direction during simulation. Current always flows from higher to lower voltage.

1.3.2 Analog simulation

Analog simulation is by far more complicated than digital circuit simulation, analog simulators use physical laws in defining analog elements models and in defining the relationship between elements. Most analog simulators create a sparse matrix, and using integrating algorithms that take variable steps in time and solve the sparse matrix for all voltages and currents for each time step. The solving of the sparse matrix involves solving a set of equations using iterative methods, such as Newton-Raphson.

As was mentioned previously in section 1.2.1 in the comparison between analog and digital simulators circuit size handling capacity for analog simulator is much more limited compared to digital simulators. This comes from the fact that the sparse matrix size is increased for each additional element added. Unlike digital simulators there are always approximations in voltage and current values, errors can accumulate for each step in time. There are good analog simulators available dealing with real circuit elements such as resistors, transistors, diodes... etc, those simulators give reliable simulation results. Classical analog simulators do not have the capability of higher levels of circuit descriptions, where digital simulators have many levels of abstraction.

In short, analog simulators use continuous signal values, the time step is finer to reduce approximation errors, signal flow is not unidirectional, number of elements affect the speed of simulation, and accuracy is important factor in the correctness of the simulation.

As a result of the need of higher level of abstraction in analog simulators as was mentioned in the previous paragraph, some analog high level description languages have evolved from the classical analog simulators. The new analog simulators cover the area between the classical analog simulators and the digital simulators. The high-level analog simulators use analog functional descriptions that are similar to digital simulators, and at the same time they use analog signal levels. One of these is spectreHDL® which uses functional description text files to model the behavior of electrical circuits [9]. The functional description includes defining the mathematical relationships between input and output terminals, their connections, and the parameters for the elements defined.

1.3.3 Mixed-signal simulation

As was mentioned in the preface in section 1.1.2 many of today's practical systems contain both analog and digital parts. It would be impractical to use analog simulators for digital circuits since it contains thousands of elements, and it would be imprecise to use digital simulators for analog circuits since it needs more accurate calculations. So a new class of

simulators appeared that can simulate mixed-signal systems. Next the different kinds of mixed-signal simulators are mentioned.

The main configurations for mixed-signal simulators are native, mixed-simulator, glued, digital with limited analog, and analog with limited digital [10]. Native configuration combines both analog and digital simulators into one simulator, mixed configuration used in a logic simulator in addition to the native simulator, glued configuration combines both an analog and digital simulator together with an interconnecting function, digital with limited analog uses digital simulator that has few analog capabilities, and the analog with limited digital is an analog simulator with few digital simulator capabilities [10].

SpectreVerilog® is a glued mixed-signal simulator. As the name implies, it can utilize both Verilog® and SpectreHDL® simulators [11]. Figure 1.4 shows the relationship between analog and digital simulators [11].

In mixed-signal simulation Interface Elements (IEs) are introduced at the transition points between the analog and the digital parts of the circuit, an analog-to-digital IE in the analog to digital direction and a digital-to-analog IE in the digital to analog direction. During a simulation, the intermediate results are transferred between the two sides. The IE elements characteristics are defined, these characteristics include rise time, fall time, delay, high voltage (logic one), low voltage (logic zero)... etc. The tool has

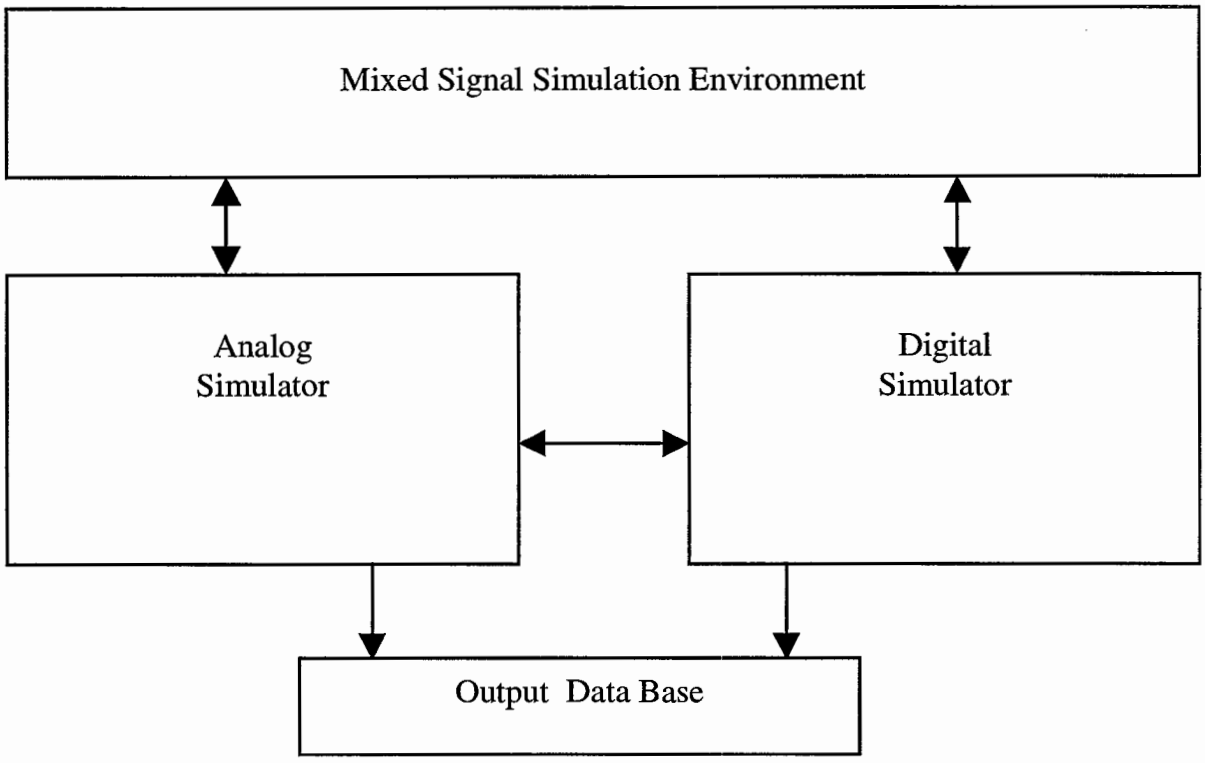


Figure1.4 Relation between analog and digitital simulators in mixed-signal simulation [8].

ready-made IEs for TTL or MOS that can be used accordingly.

1.4 Ethernet

Ethernet is a cheap, fast, general purpose networking interface. It started in the early 70s at 3Mb/s. IEEE first standardized it as the 802.3 standard in 1983, and it went into three major steps, the 10Mb/s in 1990, 100Mb/s in 1995 and finally the 1000Mb/s in 1998, where each has its own varieties of specifications that support different physical media. The physical media is the media connecting elements of the network. For more detailed historical perspective about the Ethernet see [3,4].

Ethernet uses a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) scheme to regulate access to a hardware medium, where almost an unlimited number of devices could be connected to the same cable, such that a host transmits when the cable is not in use. When two devices start transmitting at the same time, a collision occurs. Both senders then wait for a random amount of time before transmitting again, CSMA/CD algorithm is shown in figure 1.5. The packet transmitted is received by all the other hosts, and discarded by those it is not addressed to.

The 802.3z CSMA/CD standard represents the first and the second layers in the OSI (Open System Interconnection), which are the physical and data link layers shown in figure 1.6 [6].

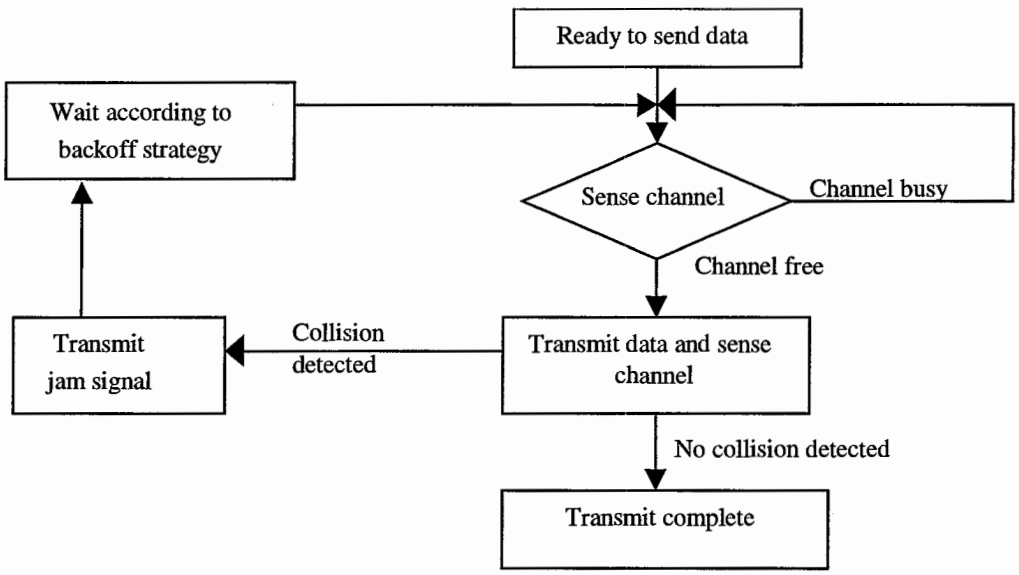


Figure 1.5 CSMA/CD algorithem.

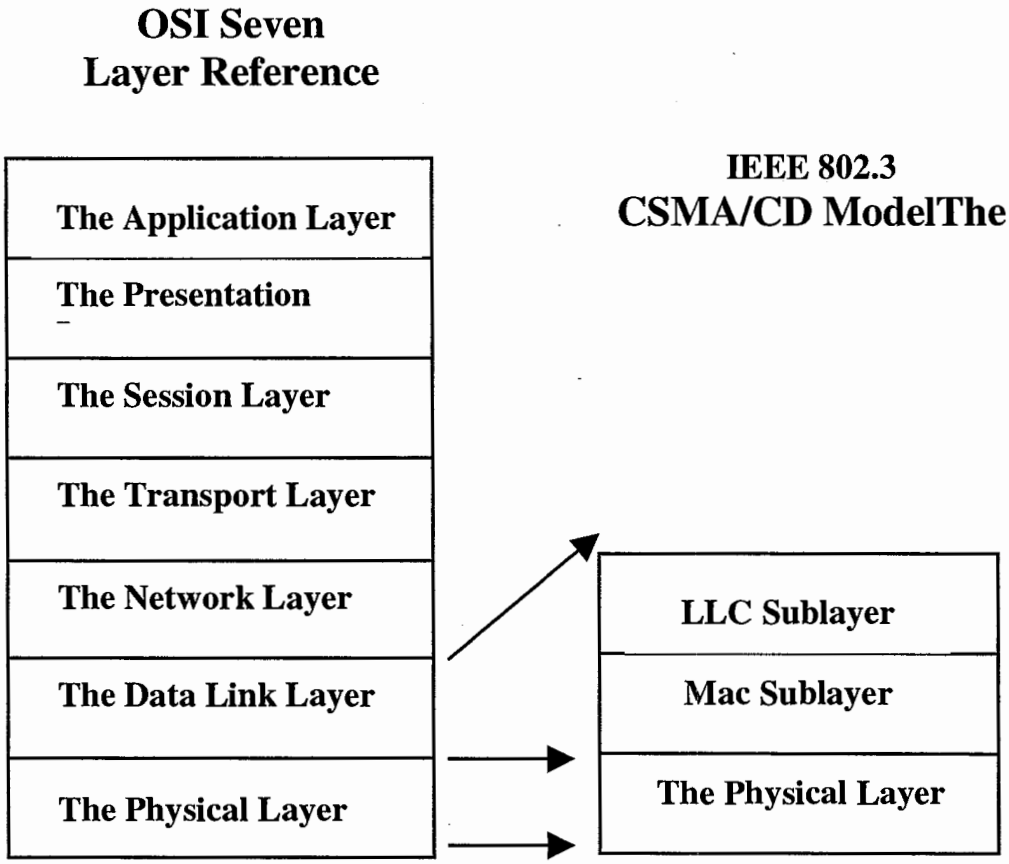


Figure 1.6 Ethernet location in the OSI stack.

The IEEE standard 802.3z (Gigabit Ethernet) supports backward compatibility with previous versions. The most important attribute is that it still uses the same frame format. Another thing that was maintained is that Ethernet continues to support an enhanced form of medium-independent interface at the interface between the MAC and PHY layers. This interface is capable of supporting the different generations and media varieties. It also has the capability of automatic link configuration through the auto-negotiation process, use of full-duplex links, and Ethernet flow control, for more information about standard 802.3z see [3,4,5,6].

1.5 Digital Synthesis

As mentioned in section 1.3.1 about digital simulation, one can write synthesizable code with little additional effort into the coding process. What that means is not every code is synthesizable. HDL can have many levels of abstraction ranging from system level to switch level. Between those two levels the RTL (Register Transfer Level) description is considered to be a synthesizable form of HDL code [11]. The RTL description specifies the data flow between registers and how data is processed by the design. Using logic synthesis tools, a gate level netlist can be generated.

A typical synthesis flow diagram is shown in figure 1.7. As shown synthesis tools need to have a library of standard gates designed with specific layout dimensions to ease the later placement and route processes. This library would have the timing information, power consumption and area about each of these gates, and it would have different versions of the same gate with different sizes, which result in different speeds, power consumption and area values. The process starts by transferring the RTL description into a gate level netlist. The next step is to map the general gate level netlist to the technology library provided. Using the specification of the inputs, outputs, and clock signals, and setting the constraints for power, timing and area, the synthesizer begins the optimization process until the design satisfies the required constraints. It would go through much iteration until it

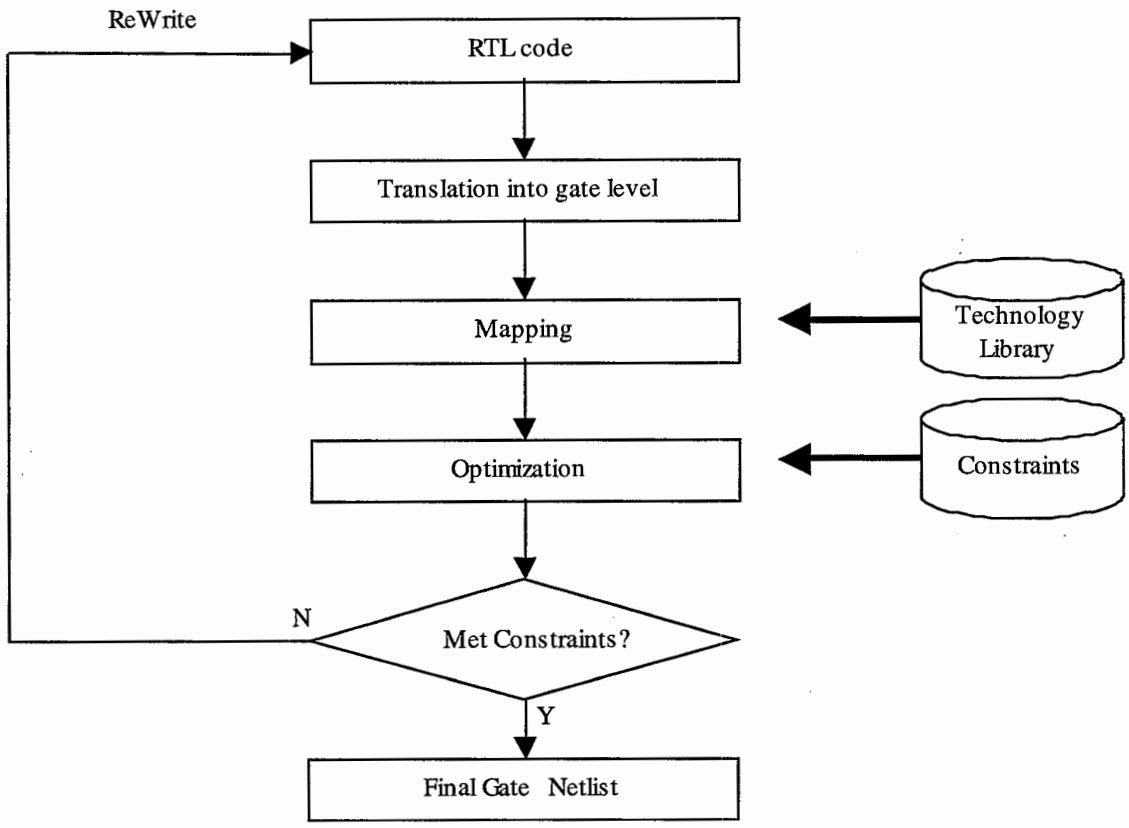


Figure 1.7 RTL synthesis flow diagram.

states the results. It should be noted that since the sizes of the designs are usually large, there is no straightforward method of optimization since that would require impractical time length.

The methods used are mostly heuristic in nature, and the results depend on the original code that it started with. The tool might not reach a good result although the design could be optimized.

The synthesis tool used in this work is the Design Compiler ® supplied by Synopsys®, known as Synopsys®, which is a powerful synthesis tool that

works on RTL code and generates gate level netlist.

Finally it is worth mentioning that several behavioral synthesis tools are commercially available [9,10], they are still limited to certain applications but the list is growing, which will allow the designers in the future to write high-level behavioral description. The tool can generate an RTL code, which then can be synthesized to get the gate level netlist [10].

1.6 Thesis Perspective

In this work, the 1000BASE-X is functionally modeled according to the IEEE 802.3z standard. The models are simulated to verify the operability of the standard and the feasibility of the project. The PCS sublayer is synthesized and a gate level netlist is achieved.

This work has a commercial value and was done in accordance with a company's specifications. The novelty of this work lies in applying the mixed-signal simulation tools to such a complex networking system as the IEEE 802.3z 1000BASE-X Standard (Gigabit Ethernet). Finally this work has an importance in the challenging process of designing mixed-signal systems.

The thesis is organized as follows. Chapter two provides the specifications and the functionality of the physical layer of the 1000BASE-X. Simulation results for he PHY are shown in chapter three. Synthesis of the PCS results and discussion are shown in chapter four. Finally the conclusion is presented in chapter five.

There are three appendices, Appendix A contains the data code for the 8b/10b coding scheme, Appendix B contains the state diagrams of the different parts of the PCS, and Appendix C contains the scripts for the synthesis.

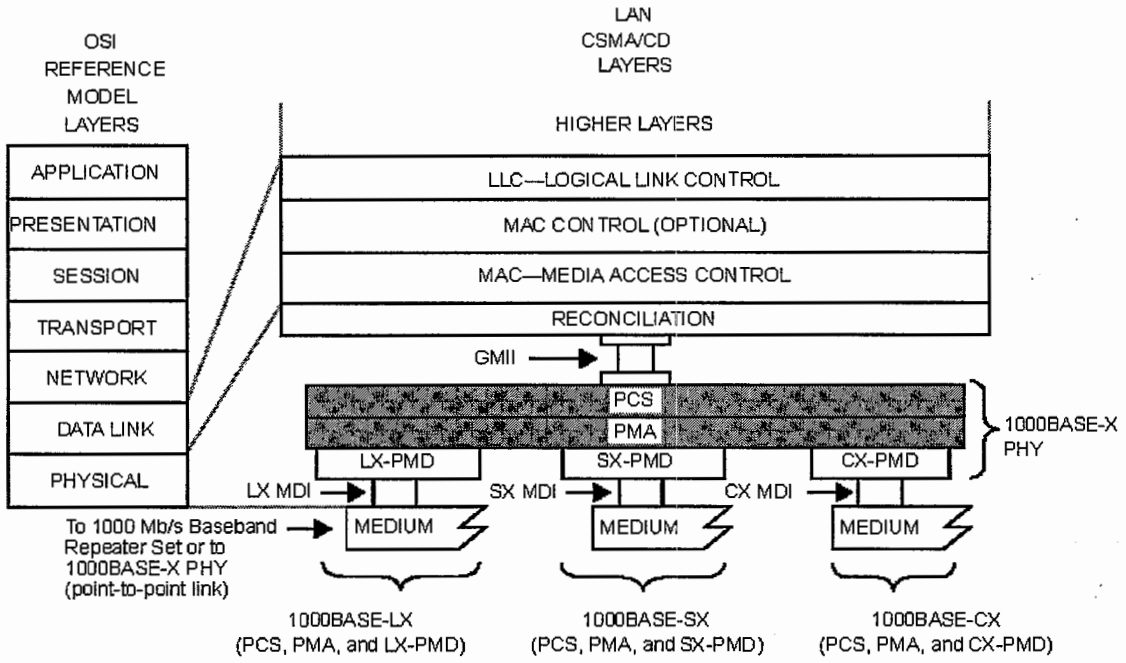
2. 1000BASE-X LAYER SPECIFICATIONS AND IMPLEMENTATION

2.1 Introduction

In this chapter the IEEE standard 802.3z (1000BASE-X) is described in general, the specifications for different layers are summarized, and the theory of operation is highlighted. The top-level interface is also described.

2.2 General Specification

The relationship between the 1000BASE-X IEEE 802.3z CSMA/CD different layers, and the corresponding Open System Interconnection (OSI) seven layers model is shown in figure2.1 [6]. This standard is targeting the lower layers of the interconnection. As shown in figure 2.1 the 1000BASE-X physical layer (PHY) has three sublayers, the physical coding sublayer (PCS), the physical medium attachment sublayer (PMA), and the physical medium dependent sublayer (PMD). As can be seen in figure 2.1, we have three types of 1000BASE-X, two for fiber and one for copper. 1000BASE-LX and 1000BASE-SX are for operation over a pair of optical fibers and long-wave length and short-wave length optical transmission, respectively. 1000BASE-CX is for operation over a single copper media (Cat-3). The family of 1000BASE-X uses the same PCS and PMA sublayers, but different PMD sublayer.



MDI MEDIUM DEPENDENT INTERFACE
GMII GIGABIT MEDIA INDEPENDENT INTERFACE
PCS PHYSICAL CODING SUBLAYER
PMA PHYSICAL MEDIUM ATTACHMENT
PHY PHYSICAL LAYER DEVICE
PMD PHYSICAL MEDIUM DEPENDENT
LX-PMD PMD FOR FIBER—LONG WAVELENGTH, clause 38
SX-PMD PMD FOR FIBER—SHORT WAVELENGTH, clause 38
CX-PMD PMD FOR 150-OHM BALANCED COPPER CABLING, clause 39

NOTE—The PMD sublayers are mutually independent.
GMII is optional.

Figure 2.1 802.3z 1000BASE-X different with respect to OSI model layers. (From IEEE Std. 802.3z-1998,©1998 All rights reserved [6])

A summarized block diagram that shows a general block functionality for the 1000BASE-X PHY is shown in figure 2.2 [6]. Figure 2.2 also shows, the top level interface signals between the gigabit media independent interface (GMII) and the PCS, the PCS main blocks, transmit, receive, auto-negotiation, carrier-sense and synchronization, the PMA main blocks transmit and receive, and the main interface signals between sublayers.

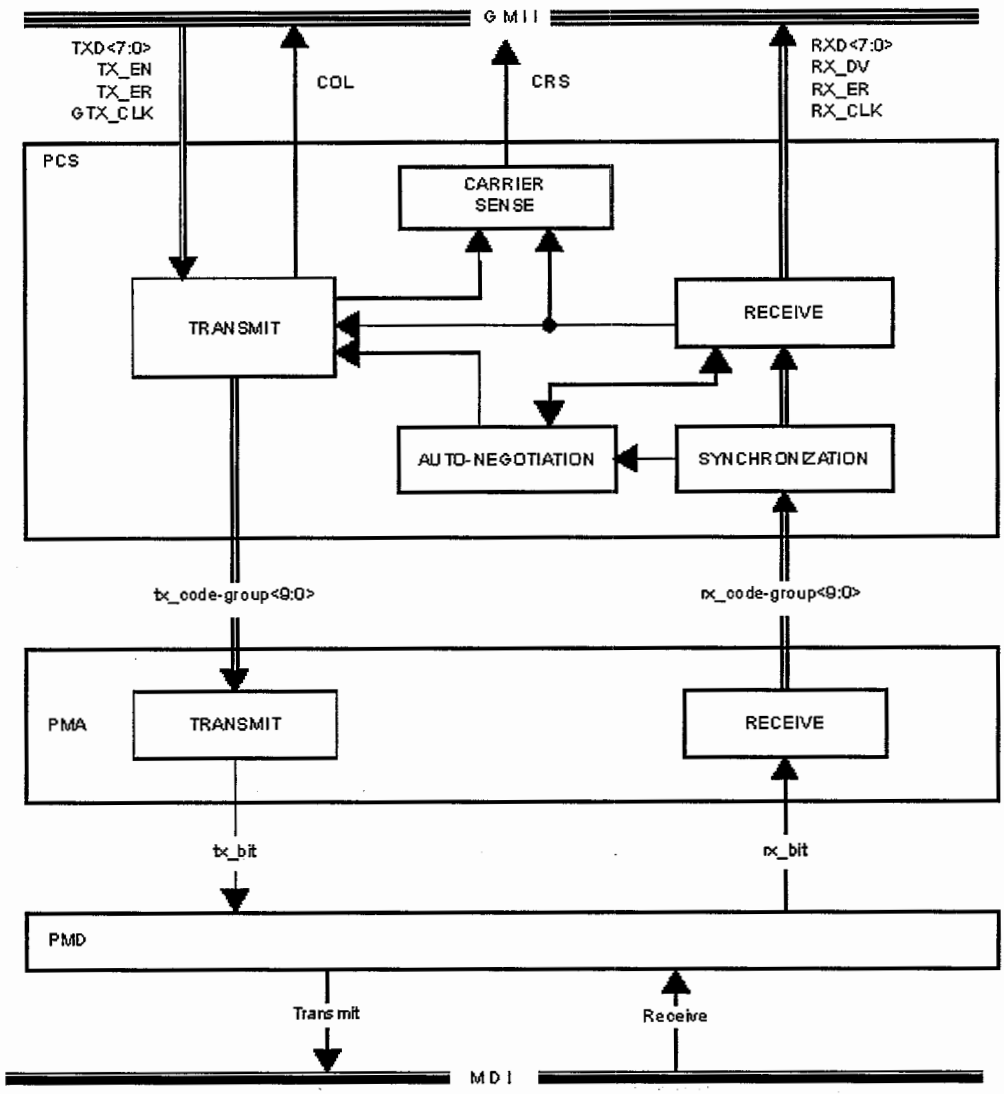


Figure 2.2 1000BASE-X functional block diagram
(From IEEE Std. 802.3z,©1998 All rights reserved.[6])

The blocks shown in figure 2.2 will be described further in the discussion of the PCS and PMA sublayers later in this chapter. In this work, we are modeling the PCS, the PMA and the management part. Although the management part is not explicitly mentioned as part of the 1000BASE-X PHY, it is implied by the standard to have a management block associated

with the PHY. A description for the interface from the top level will be provided. The implied management part will be described as well. The organization would be top down starting from the higher layer down, with the exception of the management part.

An important note here is that, since this chapter is a description for the IEEE 802.3z standard, most of what is mentioned here are contained, implied or referenced by the standard.

2.3 Gigabit Medium Independent Interface (GMII)

The function of this optional sublayer is to provide a uniform interface to the different types of the gigabit Ethernet types, which include the 1000BASE-X mentioned before and the 1000BASE-T, which defines the gigabit Ethernet across (Cat-5) physical media. Figure 2.3 shows the position of the GMII with respect to different types of the gigabit Ethernet.

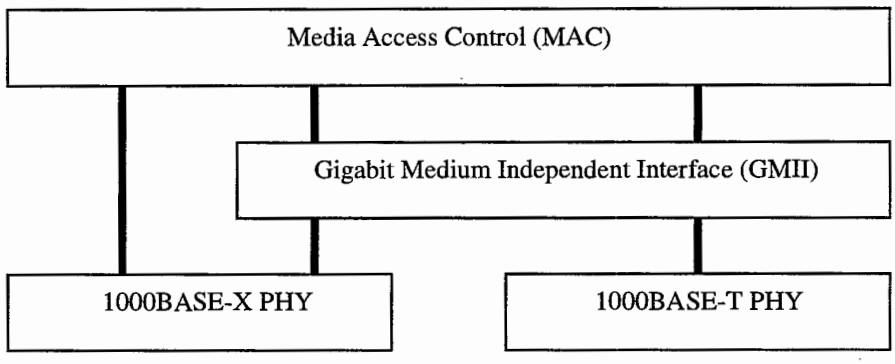


Figure 2.3 GMII position in the Ethernet layers.

The GMII sublayer is optional for the 1000BASE-X, as shown in figure 2.3, the GMII is between the MAC layer and the PHY layer reconciliation sublayer. In addition, it provides the interface for the management. This interface supports 1000 Mb/s, full duplex, independent eight-bit transmit and receive, and a simple management interface.

In this document the signals names are in italics and their values is between quotations. Figure 2.4 shows the set of signals that are exchanged between the GMII and both the MAC layer and the physical layer. At the PHY side there are four groups of signals transmit group, receive group, status group and management group. The transmit group signals are eight-bit of data *TXD<7:0>*, the enable transmission *TX_EN*, the error in transmission *TX_ER*, and the synchronizing signal *GTX_CLK*. In table 2.1

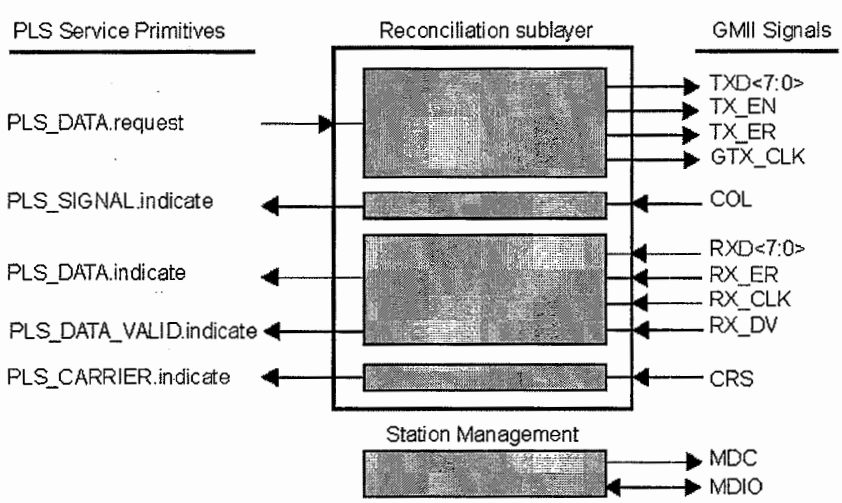


Figure 2.4 Reconciliation Sublayer (RS) inputs and outputs and STA [6] connections to GMII (From IEEE Std. 802.3z,©1998 All rights reserved).

the combinations of the signals in transmission are shown [6].

The basic frame transmission is shown in figure 2.5, where first *TX_EN* and *TX_ER* are both low to indicate idle state between inter-frame transmission, then the *TX_EN* is set to one to indicate normal data

Table 2.1 Permissible encoding of *TXD<7:0>*, *TX_EN*, and *TX_ER* [6].

TX_EN	***TX_ER***	***TXD<7:0>***	***Description***	***PLS_DATA.request parameter***
0	0	00 through FF	Normal inter-frame	TRANSMIT_COMPLETE
0	1	00 through 0E	Reserved	—
0	1	0F	Carrier Extend	EXTEND (eight-bits)
0	1	10 through 1E	Reserved	—
0	1	1F	Carrier Extend Error	EXTEND_ERROR (eight-bits)
0	1	20 through FF	Reserved	—
1	0	00 through FF	Normal data transmission	ZERO, ONE (eight-bits)
1	1	00 through FF	Transmit error propagation	No applicable parameter
NOTE—Values in TXD<7:0> column are in hexadecimal.				

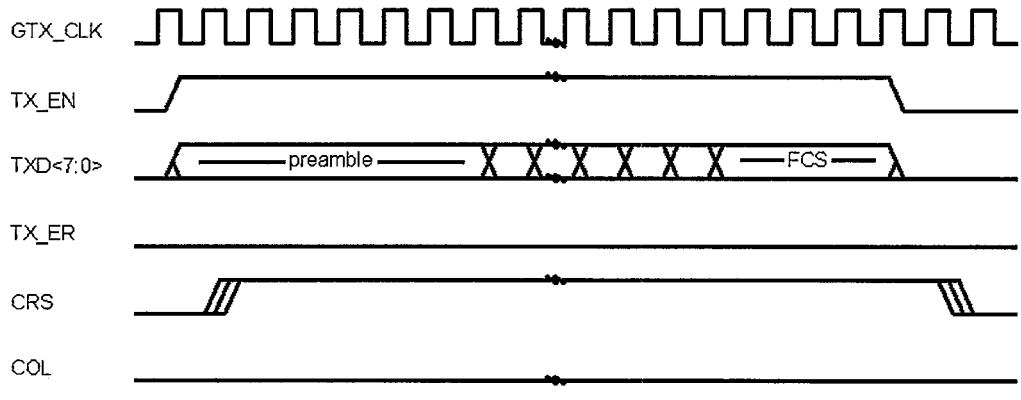


Figure 2.5 Basic frame transmission.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

transmission then reset to indicate end of transmission.

The indication of an error during transmission is shown in figure 2.6. When the *TX_ER* is set at the same time the *TX_EN* is set that indicates an error as shown in figure 2.6, *TXD* are indicated as "XX". *CRS* and *COL* signals will be described later in this section.

The transmission with carrier extension is shown in figure 2.7, burst

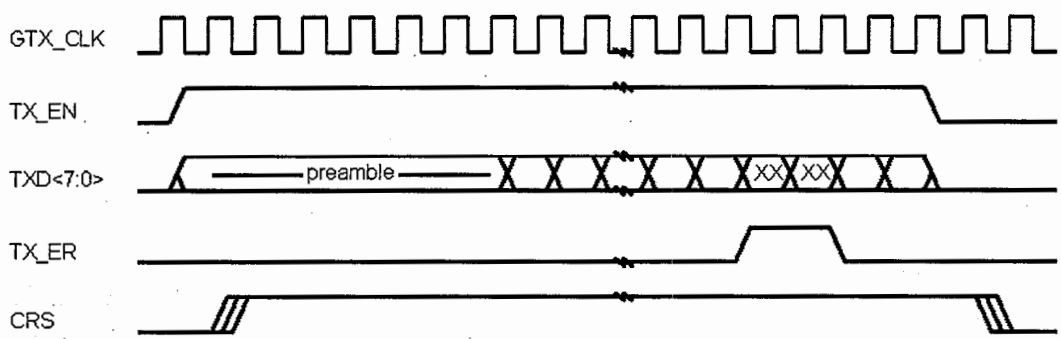


Figure 2.6 Propagation an error within a frame.
(From IEEE Std. 802.3z,©1998 All rights reserved [4])

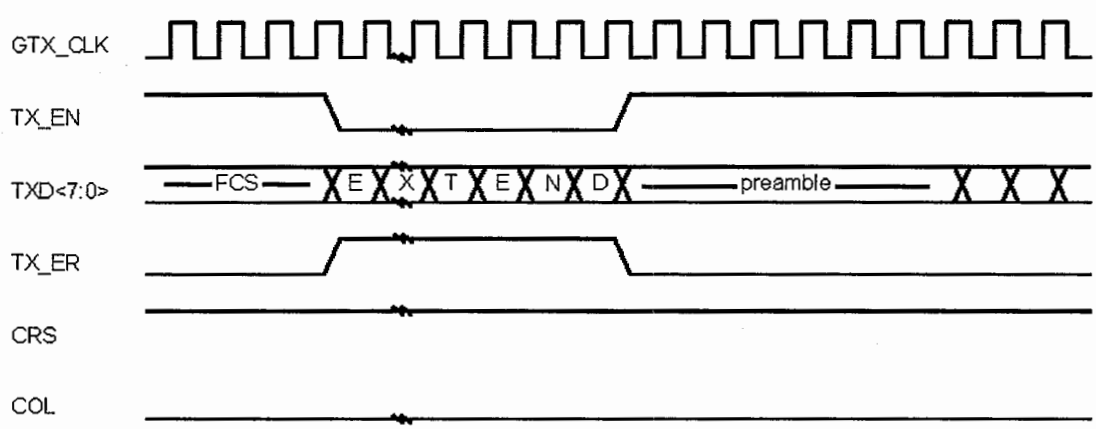


Figure 2.7 Burst transmission.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

transmission indicate multiple packet transmission without giving up the interface. Setting the *TX_ER*, resetting the *TX_EN* and placing the value hexadecimal "0F" on the *TXD* data lines indicates the packet extension, as shown in figure 2.7. Not shown is the indication of an error during extension [6], where the TXD is set to hexadecimal "1F" during extension.

Receive signals receive eight-bit of data *RXD*, receive data valid *RX_DV*, receive error *RX_ER*, and the synchronizing signal *RX_CLK*. Table 2.2 shows the combinations of these signals used. As in the transmit signals, there are some of the reserved cases for future use.

The basic frame reception is shown in figure 2.8. When both *RX_DV* and *RX_ER* are low it indicates inter-frame idle, setting the *RX_DV* indicates the data reception on the *RXD* lines, resetting the *RX_DV* indicates end of frame.

The frame reception with carrier extension is shown in figure 2.9. Setting the *RX_ER*, resetting the *RX_DV* and setting the *RXD* to hexadecimal "0F" indicates frame extension.

Similar to the previous case burst reception is shown in figure 2.10, where the signal *RX_ER* is set during extension, resetting the *RX_DV* and setting *RXD* to hexadecimal "0F" value, between frames transmitted.

The status signals are the carrier sense *CRS* and the collision *COL*. The *CRS* signal sets to one to indicate either transmit or receive.

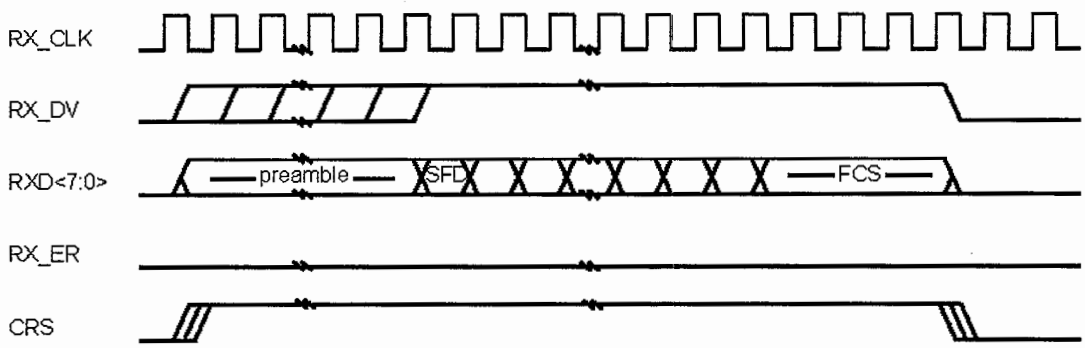


Figure 2.8 Basic frame reception.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

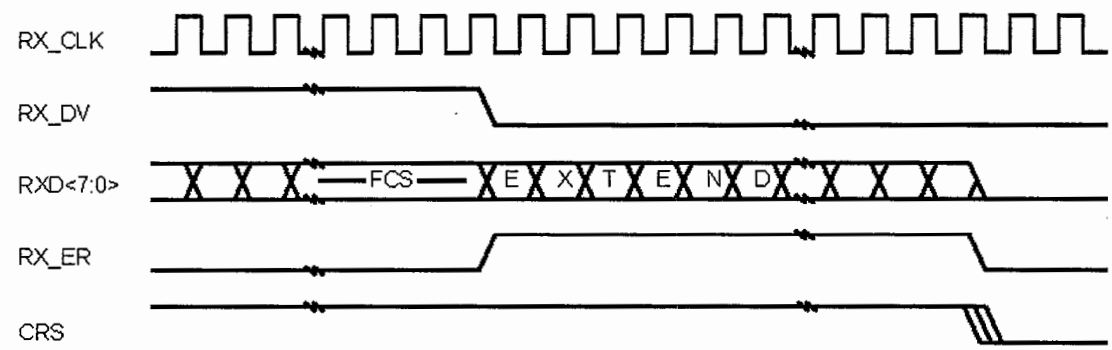


Figure 2.9 Frame reception with carrier extension.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

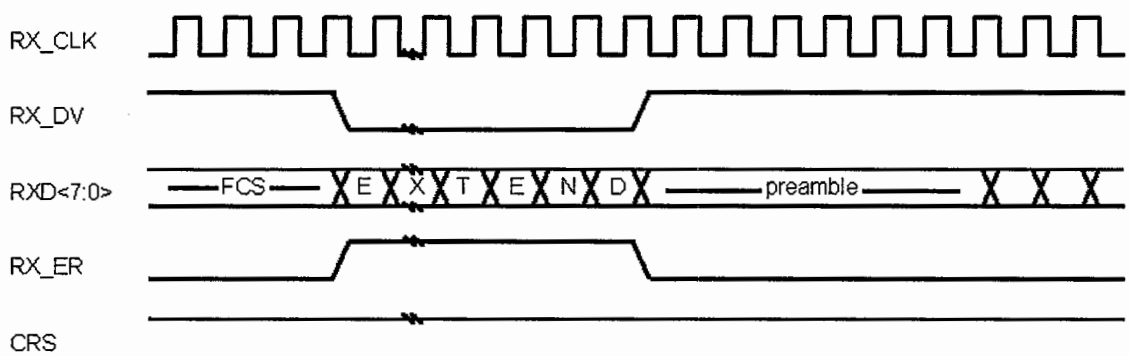


Figure 2.10 Burst reception.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

Table 2.2 Permissible encoding of *RXD<7:0>*, *RX_ER*, and *RX_DV* [6].

RX_DV	*RX_ER*	*RXD<7:0>*	Description	PLS_DATA.indicate parameter
0	0	00 through FF	Normal inter-frame	No applicable parameter
0	1	00	Normal inter-frame	No applicable parameter
0	1	01 through 0D	Reserved	—
0	1	0E	False Carrier indication	No applicable parameter
0	1	0F	Carrier Extend	EXTEND (eight-bits)
0	1	10 through 1E	Reserved	—
0	1	1F	Carrier Extend Error	ZERO, ONE (eight-bits)
0	1	20 through FF	Reserved	—
1	0	00 through FF	Normal data reception	ZERO, ONE (eight-bits)
1	1	00 through FF	Data reception error	ZERO, ONE (eight-bits)
NOTE—Values in RXD<7:0> column are in hexadecimal.				

The *COL* signal is set to one to indicate the detection of a collision on the medium, that is transmission and reception at the same time. All the previous figures 2.5-2.11 show the behavior of the CRS signal. Figure 2.11 shows an example of transmission with collision.

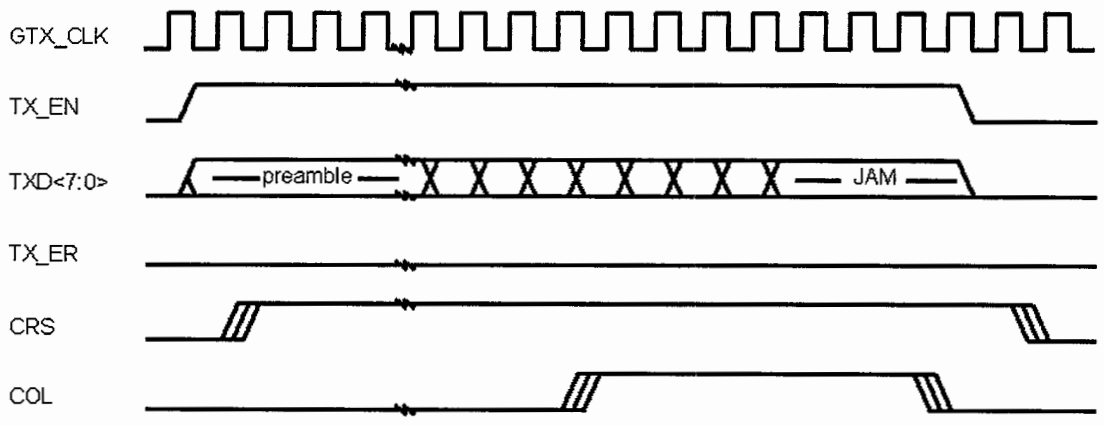


Figure 2.11 Transmission with collision.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

The management signals, management data input output *MDIO* and management data clock *MDC*, are used to exchange control and configuration data between the management entity and the physical layer. The data is exchanged using a specific frame format, where the data is transferred serially along the *MDIO* line synchronized with *MDC*. The frame format is shown in table 2.3 [11].

The data is applied serially from the left to right across the *MDIO* line, synchronized with the MDC clock. The fields are:

1. *PRE* (preamble), this is a thirty two-bit field of consecutive 1s on MDIO, to indicate beginning of the management frame, this field is optional.

2. *ST* (start delimiter), this is a two-bit field. A (01) transition indicates the start of the management frame.

3. *OP* (operation code), this is a two-bit field. A (01) combination is for write and (10) combination is for read.

4. *PHYAD* (PHY address), this is a five-bit address field. Selects which PHY should respond on multiple PHY system.

5. *REGAD* (register address), this is a five-bit field. Selects which

Table 2.3 Management frame Structure [6].

	Management Frame fields							
	PRE	*ST*	*OP*	*PHYAD*	*REGAD*	*TA*	*DATA*	IDLE
Read	1...1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
Write	1...1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDD	Z

register is to be operated on.

6. *TA* (turn around), this is a two-bit field. Allows the line to turn around from being driven by the management entity to being driven by the PHY during the read operation.

7. *DATA* (frame data) is a sixteen-bit data field.

8. IDLE is the inter-frame spacing, where none is driving MDIO line.

2.4 Physical Coding Sublayer (PCS)

The functional block diagram for the 1000BASE-X PHY was shown in figure 2.4, where it shows the interface signals with the GMII from top level and with the PMD in the lower level. It also shows the internal block diagram and the interconnections between different sub blocks for the PCS layer. The main functions within the PCS are: PCS transmit; carrier sense; synchronization; PCS receive; and auto-negotiation as shown on figure 2.2.

During the transmit process, the PCS maps the *TXD* eight-bit data interface with the GMII into a ten-bit code-group down to the PMA. And during the receive process, the PCS maps the ten-bit into the *RXD* eight-bit data interface with the GMII. The mapping process is done using an 8b/10b coding scheme. Encoding is enclosed within the transmit block, while decoding is enclosed within the receive block.

The functions of the blocks within the PCS, and the way they were implemented are discussed next. A separate section is devoted for the 8b/10b

coding system used. The functionality of the PCS is also described in section 2.5. Note that all the state machines used are in Appendix B.

2.4.1 Transmit

The data is presented to the PCS through the *TXD* eight-bit bus width, framed using the *TX_EN* and *TX_ER* signals. The type of the code system to be used is based on the units of the code-groups. There are two types of code-groups; data code-groups and special code-groups. The data code-groups are mapped one to one from the input data *TXD*. The special code-groups are used for signaling purposes. The signaling is used between peers on the PHY level, the signaling purposes are; indicating the start, end, extension of packets, transmitting auto-negotiation information, and to keep the connection during the idle time. During transmission of the signaling information it is mapped into code-groups where it can have from one to four code-groups.

The transmit function has two parts, the first is called the order-set generator, and the second is the code-group generator. In the order-set generator the type of the order-set to be transmitted is determined according to the state of the connection as determined by the auto-negotiation block, and it also determined by the transmit inputs *TX_EN*, *TX_ER* and *TXD* from the top level.

In the code-group generator, the code-groups are simply generated

according to the order-sets.

In the standard, the order-set generator is defined using a finite state machine (FSM) (Appendix B) [6]. This state machine, during packet transmission, is driven by the GMII transmit signals and it uses the *GTX_CLK* as the synchronizing signal.

The code-group generator is also defined using an FSM [6]. It uses the signals from the order-set generator to determine the code-groups to be transmitted. The code-groups generated are eight-bit wide, so they are encoded through the 8b/10b encoder.

These two state machines were implemented in Verilog®HDL at the RTL level of abstraction, then they are tested and verified at the block level. The whole transmit block is formed by combining these two sub blocks. The results for the testing are shown in chapter three.

2.4.2 Receive

The receive block reverse the function of the transmit block, it takes the encoded code-groups received by the synchronization block and generates the corresponding *RXD<7:0>*, *RX_ER*, and *RX_DV* signals to the GMII. The *RX_CLK* signal is also passed to the GMII. This block receives the code-groups and determines the corresponding signals values that generated them. It uses the 8b/10b decoder to decode the code-groups. The receiver uses the special code-groups to determine the packets boundary, the auto-

negotiation results, and the idle connection state. The receiver functionality is defined using an FSM (Appendix B) [6]. The receiver state machine was implemented using Verilog®HDL RTL level, the code was tested and verified at the block level. The results for the testing are shown in chapter three.

2.4.3 Synchronization

The data is transmitted along the channel serially, so the boundaries between the code-groups should be attained. The function of this block is to ensure that the code-group boundaries are locked, and pass the received code-groups to the receive block. To ensure that the PMA is working correctly, three consecutive comma code-groups should be received. The functionality of this block is defined using an FSM (Appendix B) [6]. The synchronization was built using Verilog®HDL at the RTL level, the code was tested and verified. The results for the testing are shown in chapter three.

2.4.4 Auto-negotiation

The function of the auto-negotiation block is to determine the capabilities of the remote-link partner, advertise local devise capabilities, and determine the optimum common mode of operation. During the auto-negotiation phase the transmit and the receive blocks are controlled by the auto-negotiation process. As for other blocks, the functionality was defined in an FSM format [6].

The auto-negotiation was built using Verilog® HDL at the RTL level,

the code was tested and verified. The results for the testing are shown in chapter three.

2.4.5 Carrier sense

The function of the carrier sense block is to monitor the transmitted and the received data packets' activities. Depending on whether the PCS is implemented as a repeater or as Data Terminal Equipment (DTE), the *CRS* signal would be asserted. For a repeater it would be asserted when a packet is received. For DTE, CRS is asserted when either transmit or receive is observed. The carrier sense block is defined as an FSM (Appendix B) [6]. The carrier sense block was built using Verilog®HDL and code is written at the RTL level, the code was tested and verified. The testing results are shown in chapter three.

2.4.6 Management

The management block is not explicitly defined. It is a collection of registers that are used in the PCS. It uses the frame format that was mentioned in table 2.3. In the standard there is no explicit format for the management block except for the management frame format and the registers it contains. The management registers are shown in table 2.4. As shown in the table we have 32 register address space, where registers 0 through 8 and register 15 are used by the 1000BASE-X PHY, others are either used by other PHY implementations or reserved. Note in the third

field the registers are described as basic or extended. The extended registers are optional for the standard, there are three basic registers only.

Following is a brief description for the basic registers [6]. Table 2.5 shows the contents of Control register (register 0), the Control register controls the modes of operation of the PHY, and all of the bits can be read from and write to.

Table 2.6 shows the contents for the Status register (register 1), the Status register summarizes the current state of the PHY, and all of the bits are read only. Finally, table 2.7 shows a description for the Extended Status register (register 15). As the name imply the Extended Status register is an extension for the Status register, and all the bits in the Extended Status

Table 2.4 Management register set [6].

Register address	Register name	Basic/ Extended
0	Control	B
1	Status	B
2,3	PHY Identifier	E
4	Auto-Negotiation Advertisement	E
5	Auto-Negotiation Link Partner Base Page Ability	E
6	Auto-Negotiation Expansion	E
7	Auto-Negotiation Next Page Transmit	E
8	Auto-Negotiation Link Partner Received Next Page	E
9	100BASE-T2 Control Register	E
10	100BASE-T2 Status Register	E
11 through 14	Reserved	E
15	Extended Status	B
16 through 31	Vendor Specific	E

register are read only

The extended registers are optional, following is a general description for the extended registers. The PHY Identifier registers (registers 2 and 3), contain identification; company's unique identifier number, model and revision of the PHY implementation

Table 2.5 Control register bit definitions [6].

Bit(s)	Name	Description	R/W[a]
0.15	Reset	1 = PHY reset 0 = normal operation	R/W SC
0.14	Loopback	1 = enable loopback mode 0 = disable loopback mode	R/W
0.13	Speed Selection (LSB) 0.6 0.13	1 1 = Reserved 1 0 = 1000 Mb/s 0 1 = 100 Mb/s 0 0 = 10 Mb/s	R/W
0.12	Enable Auto-Negotiation	1 = Enable Auto-Negotiation Process 0 = Disable Auto-Negotiation Process	R/W
0.11	Power Down	1 = power down 0 = normal operation[b]	R/W
0.10	Isolate	1 = electrically Isolate PHY from GMII 0 = normal operation[b]	R/W
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process 0 = normal operation	R/W SC
0.8	Duplex Mode	1 = Full Duplex 0 = Half Duplex	R/W
0.7	Collision Test	1 = enable COL signal test 0 = disable COL signal test	R/W
0.6	Speed Selection (MSB) 0.6 0.13	1 1 = Reserved 1 0 = 1000 Mb/s 0 1 = 100 Mb/s 0 0 = 10 Mb/s	R/W
0.65:0	Reserved	Write as 0, ignore on Read	R/W

a. R/W = Read/Write, SC = Self-Clearing.
b. For normal operation, both 0.10 and 0.11 must be cleared to zero.

Table 2.6 Status register bit definition [6].

Bit(s)	Name	Description	R/W[a]
1.15	100BASE-T4	1 = PHY able to perform 100BASE-T4 0 = PHY not able to perform 100BASE-T4	RO
1.14	100BASE-X Full Duplex	1 = PHY able to perform full-duplex 100BASE-X 0 = PHY not able to perform full-duplex 100BASE-X	RO
1.13	100BASE-X Half Duplex	1 = PHY able to perform half-duplex 100BASE-X 0 = PHY not able to perform half-duplex 100BASE-X	RO
1.12	10 Mb/s Full Duplex	1 = PHY able to operate at 10 Mb/s in full-duplex mode 0 = PHY not able to operate at 10 Mb/s in full-duplex mode	RO
1.11	10 Mb/s Half Duplex	1 = PHY able to operate at 10 Mb/s in half-duplex mode 0 = PHY not able to operate at 10 Mb/s in half-duplex mode	RO
1.10	100BASE-T2 Full Duplex	1 = PHY able to perform full-duplex 100BASE-T2 0 = PHY not able to perform full-duplex 100BASE-T2	RO
1.9	100BASE-T2 Half Duplex	1 = PHY able to perform half-duplex 100BASE-T2 0 = PHY not able to perform half-duplex 100BASE-T2	RO
1.8	Extended Status	1 = Extended status information in register 15 0 = No extended status information in register 15	RO
1.8:7	Reserved	ignore when read	RO
1.6	MF Preamble Suppression	1 = PHY will accept management frames with preamble suppressed. 0 = PHY will not accept management frames with preamble suppressed.	RO
1.5	Auto-Negotiation Complete	1 = Auto-Negotiation process completed 0 = Auto-Negotiation process not completed	RO
1.4	Remote Fault	1 = remote fault condition detected 0 = no remote fault condition detected	RO/ LH
1.3	Auto-Negotiation Ability	1 = PHY is able to perform Auto-Negotiation 0 = PHY is not able to perform Auto-Negotiation	RO
1.2	Link Status	1 = link is up 0 = link is down	RO/ LL
1.1	Jabber Detect	1 = jabber condition detected 0 = no jabber condition detected	RO/ LH
1.0	Extended Capability	1 = extended register capabilities 0 = basic register set capabilities only	RO

a. RO = Read Only, LL = Latching Low, LH = Latching High

Table 2.7 Extended Status register bit definition [6].

Bit(s)	Name	Description	R/W[a]
15.15	1000BASE-X Full Duplex	1 = PHY able to perform full-duplex 1000BASE-X 0 = PHY not able to perform full-duplex 1000BASE-X	RO
15.14	1000BASE-X Half Duplex	1 = PHY able to perform half-duplex 1000BASE-X 0 = PHY not able to perform half-duplex 1000BASE-X	RO
15.13	1000BASE-T Full Duplex [b]	1 = PHY able to perform full-duplex 1000BASE-T 0 = PHY not able to perform full-duplex 1000BASE-T	RO
15.12	1000BASE-T Half Duplex[b]	1 = PHY able to perform half-duplex 1000BASE-T 0 = PHY not able to perform half-duplex 1000BASE-T	RO
15.11:0	Reserved	ignore when read	RO

a. RO = Read Only.
b. b. Specifications for 1000BASE-T operation are planned for future work.

The Auto-Negotiation Advertisement register (register 4), contains the local PHY capabilities (these type of duplex, Pause bits, remote fault and next page capabilities), during auto-negotiation procedure this information in the advertisement register is to be sent to the link partner [6]. The received advertisement information from the link partner are saved in the Auto-Negotiation Link Partner Ability register (register 5).

The Auto-Negotiation Expansion register (register 6) has controls for next page capabilities. The Auto-Negotiation Next Page Transmit register (register 7), contains the next page in an additional page transmission to the link partner. The next page is used to enable the exchange of user or application specific data. The Auto-Negotiation Link Partner Received Next

Page register (register 8) is used to save the (next page) received from the link partner in a multi-page transmission during the auto-negotiation procedure.

The management block can be integrated within the auto-negotiation block but it was built at the top level of the PCS. The management block was implemented using Verilog®HDL at the RTL level, the code is tested and verified. The results for the testing are shown in chapter three.

2.4.7 PCS functionality

In this section the operation of the PCS is described. After power on, after reset, or upon request from management the PCS auto-negotiation procedure is activated. When the auto-negotiation procedure is activated it controls the transmit block such that it would start sending the configuration signaling across the channel. The receive block automatically would recognize the configuration signaling when the link partner starts sending his configuration. The configuration received is send to the auto-negotiation.

The process of auto-negotiation compares the received results with its ability and determines the connection setup according to a predefined priority resolution function. The configuration process goes through many stages until the auto-negotiation block determines that the connection has been established. The transmit function would be able to transmit data coming from top level after the connection is established, and by the same token the

transmit block of the link partner would behave the same. There is no data transmission or reception until the connection is established.

The synchronization block needs to be synchronized to the input before the auto-negotiation procedure is activated.

When the transmit function is not transmitting data it would transmit idle signaling to keep the connection alive and ready, this idle signaling helps in keeping the two link partners synchronized.

2.5 8b/10b Code

This is a data encoding and decoding scheme patented by IBM®[15]. This code is used in fiber channel for its many qualities:

- It is a balanced dc voltage code, no dc value for the code, number of 0's sent equal number of 1's.
- The code has good transition density and run length limited, which enables clock recovery.
- It has an error detection capability (disparity control), an added redundancy bits helps in error detection.
- Frame delimiting with a special bit sequence is used for this purpose (comma).

For the previous reasons this coding scheme is good for transmission over optical and copper media, so it was adopted for the 1000BASE-X signaling. There are two types of code-groups in this code, data code-groups

and special code-groups. Data code-groups result from encoding all the possible combinations of the eight-bit transmitted data as shown in Appendix A. Special code-groups are introduced by the PCS for signaling purposes. The Special code-groups are shown in table 2.7.

For all the code-groups used in this code, disparity control is defined to enhance error detection and maintain a dc-balanced code. There are some rules to compute the running disparity at the end of transmitting or receiving a code [6].

Table 2.7 Valid special code-groups.

Code Group Name	**Octet Value**	**Octet Bits** HGF EDCBA	**Current RD -** abcdei fghj	**Current RD +** abcdei fghj	**Notes**
K28.0	1C	000 11100	001111 0100	110000 1011	1
K28.1	3C	001 11100	001111 1001	110000 0110	1,2
K28.2	5C	010 11100	001111 0101	110000 1010	1
K28.3	7C	011 11100	001111 0011	110000 1100	1
K28.4	9C	100 11100	001111 0010	110000 1101	1
K28.5	BC	101 11100	001111 1010	110000 0101	2
K28.6	DC	110 11100	001111 0110	110000 1001	1
K28.7	FC	111 11100	001111 1000	110000 0111	1,2
K23.7	F7	111 10111	111010 1000	000101 0111	
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	

RD is the running disparity.
NOTES
1. Reserved.
2. Contains a comma.

The comma is the special code-group K28.5 table 2.7, it has the special characteristic of having five consecutive 1s or 0s depending on the running disparity. In 1000BASE-X this is the only allowed code-group that has the comma inside it. The comma is used in aligning the code-groups when the data is received serially

As mentioned in the transmit section the code-groups are combined into order-sets. The order-sets combinations are shown in table 2.8, each order-set is composed of one to four code-groups. The configuration order-set /C/ is used during the auto-negotiation procedure to exchange the advertisement ability registers. The idle /I/ is used when the connection is

Table 2.8 Defined ordered-sets.

Code	Ordered-Set	Number of Code-Groups	Encoding
/C/	Configuration		Alternating /C1/ and /C2/
/C1/	Configuration 1	4	/K28.5/D21.5/Config_Reg[a]
/C2/	Configuration 2	4	/K28.5/D2.2/Config_Reg[a]
/I/	IDLE		Correcting /I1/, Preserving /I2/
/I1/	IDLE 1	2	/K28.5/D5.6/
/I2/	IDLE 2	2	/K28.5/D16.2/
	Encapsulation		
/R/	Carrier_Extend	1	/K23.7/
/S/	Start_of_Packet	1	/K27.7/
/T/	End_of_Packet	1	/K29.7/
/V/	Error_Propagation	1	/K30.7/
a. Two data code-groups representing the Config_Reg value			

idle to keep the connection active. The comma is found in the /C/ and /I/ order-sets, to make sure of the code-group alignment during configuration and when the connection is idle. The special code-groups are used only for signaling between the PHYs at the connection ends, and they will not be transferred to the higher level layers.

The block diagram for the 8b/10b encoder is shown in figure 2.12 [15]. The encoder and decoder were built using blocks from 5b/6b code and 3b/4b code to form the 8b/10b code. From figure 2.12, the signal Data/special code tells the circuit if it is to generate data or special code-group. Both the 8b/10b encoder and 8b/10b decoder were implemented as a combinational circuit,

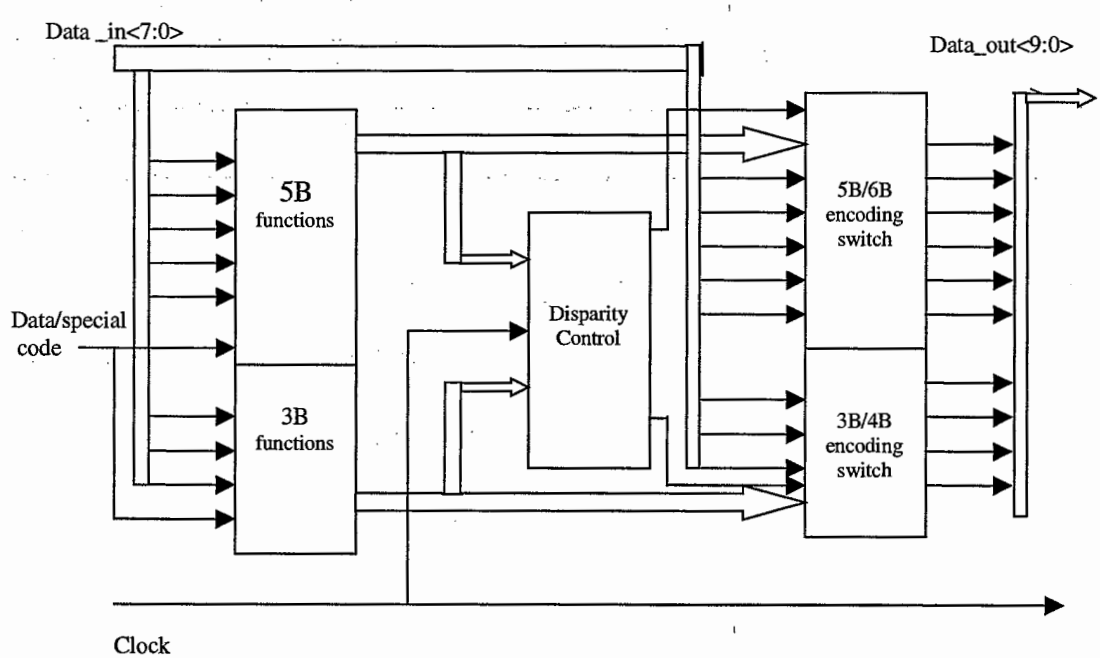


Figure 2.12 Block diagram for 8b/10b encoder [15].

with a single flip-flop to hold the disparity value for the next cycle. To accommodate for circuit delay it was assumed that there would be a clock cycle between setting the input lines and getting the coded/encoded result.

The possible set of code-groups is shown in Appendix A. The special code-groups are shown in table 2.7.

The encoder and decoder were written in Verilog® HDL at the RTL level, and it was tested and verified for all the possible input combinations for both the 8b/10b encoder and decoder. The results are not shown but in chapter three we will show the code-groups in both the encoder and decoder.

2.6 Physical Medium Attachment (PMA)

This layer takes a ten-bit from the transmit block of the PCS as input and through a parallel to serial circuit it converts it into one-bit serial output on the transmit side, and it takes a single bit serial input on the receive side, and through serial to parallel it converts it into ten-bit that is delivered to the PCS. Figure 2.13 shows a functional block diagram for the PMA.

This circuit contains analog parts to generate the required clocks for serialization and deserialization. The serialization clock is generated using a clock multiplier circuit. The clock for deserializtion is recovered from the received data through a clock recovery circuit. There are also other simple digital circuits. These include shift registers and comma detection (alignment) circuit. The clocks generation circuits are discussed next.

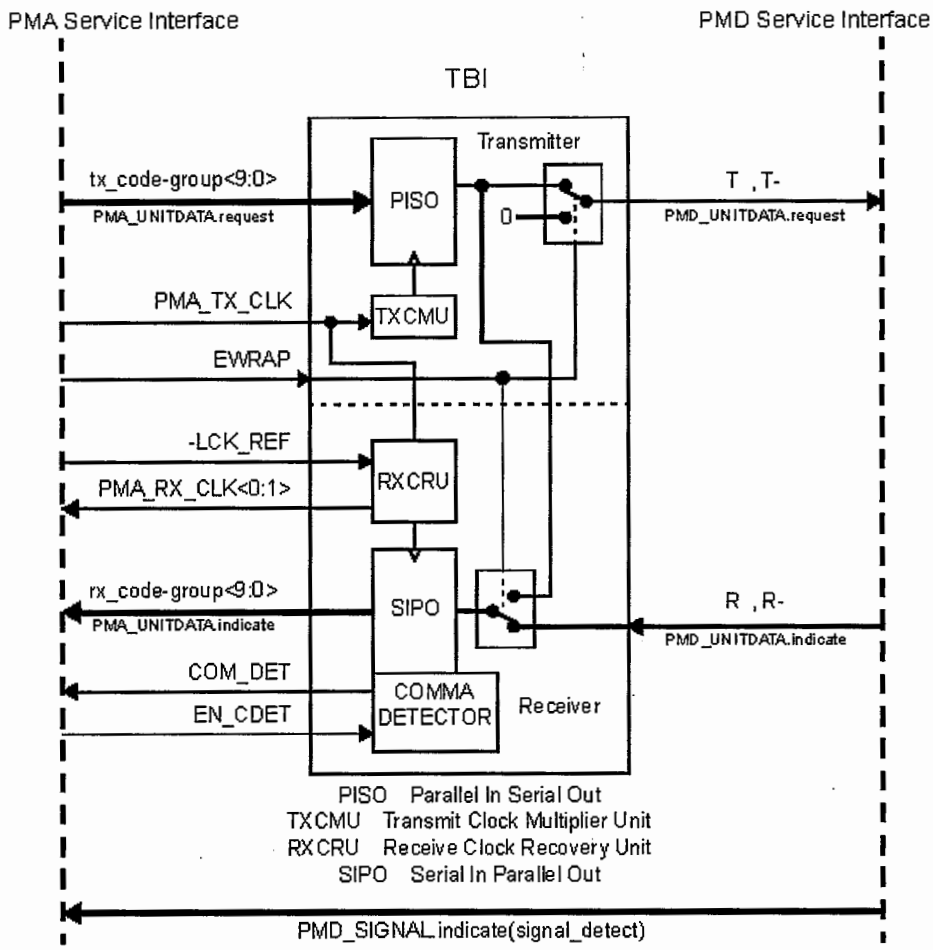


Figure 2.13 Block diagram for the PMA.
(From IEEE Std. 802.3z,©1998 All rights reserved [6])

2.6.1 Clock Recovery Circuit

The clock recovery circuit block diagram is shown in figure 2.14. This is a typical phase lock loop (PLL) circuit. In this circuit the Voltage Controlled Oscillator (VCO), Phase Detector (PD) and Comparator were implemented using SpectreHDL® language, other parts were implemented using Verilog®HDL. The simulation was done using correct 8b/10b random

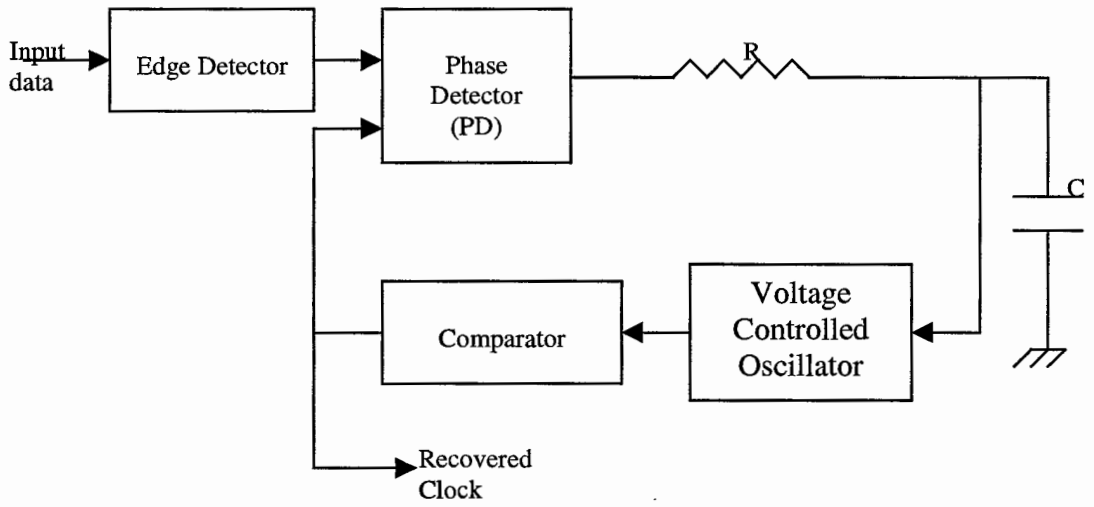


Figure 2.14 Block diagram for clock recovery circuit.

data input, the results are shown in chapter three with the PMA simulation.

2.6.2 Clock Multiplier

Similar to clock recovery circuit, this also was implemented using a PLL circuit. Figure 2.15 shows block diagram for the clock multiplier. The VCO, PD, and Comparator were implemented using SpectreHDL®. Other parts were implemented using Verilog®HDL. The circuit was simulated and verified. The results are shown next chapter.

2.7 Chapter Conclusion

In this chapter the 1000BASE-X PHY was introduced and its functionality was described. The coding languages for the digital and analog parts were highlighted. Most of the information mentioned in this chapter is

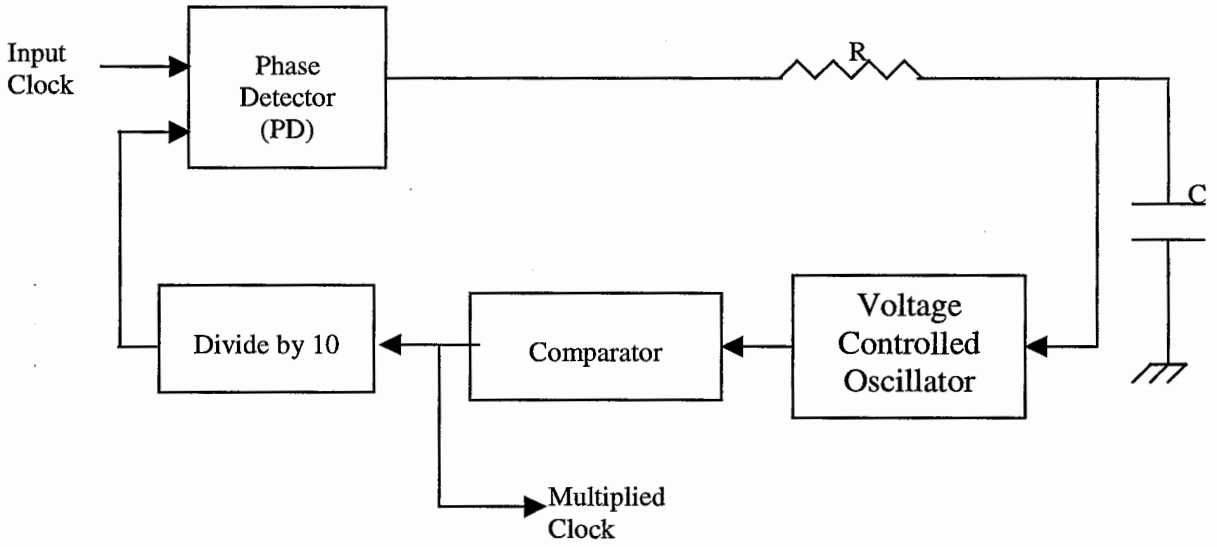


Figure 2.15 Block diagram for clock multiplier circuit.

derived, written or implied in the IEEE standard 802.3z for the 1000BASE-X.

3. 1000BASE-X LAYER SIMULATIONS AND RESULTS

3.1 Introduction

In this chapter, results from the testing for the PCS and PMA sublayers are presented and discussed. A description of the main internal signals is also given. The digital parts were written in Verilog®HDL, and the analog parts in spectreHDL®. The mixed-signal simulation was done using SpectreVerilog® simulator, where digital parts are simulated using Verilog-XL® and the analog parts using spectreHDL®. The interface elements (IE) between analog and digital were defined to be transistor-to-transistor logic (TTL) voltage levels.

As a convention in this chapter the signal names are shown in italics, their values are shown between quotations

3.2 PCS Simulations and Results

The hierarchy of the PCS is shown in figure 3.1. All the modules of the PCS are digital. They were built in Verilog®HDL. Higher order nodes were built schematically. The schematic for the PCS is shown in figure 3.2. The blocks shown in both figures 3.1 and 3.2, which are not in the functional block of figure 2.2 are added to the block diagram as follows.

The 8b/10b decoder and encoder were implicitly mentioned as part of the transmit and the receive functions in the block diagram of the PCS. The

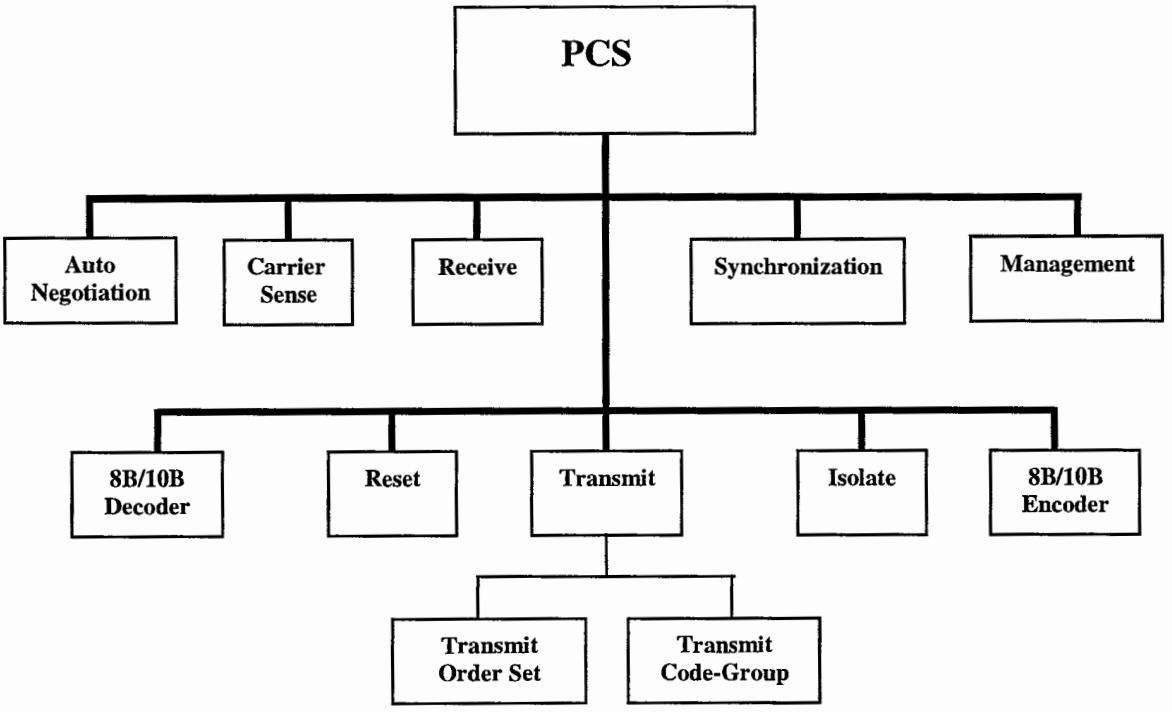


Figure 3.1 Hierarchy of the top level PCS.

management is implied by the interface signals MDIO and MDC. The isolate block is implemented at the top level to accomplish the isolate function as defined in the management control register [6]. The reset module is implemented to combine different reset signals at the top level.

Table 3.1 shows the equivalency between the names in schematic and hierarchical figures 3.1 and 3.2. Note that the schematic names are shown on the top right corner of each module in figure 3.2. In the next sections, the simulation results for the main functional and the top-level blocks are shown.

The internal signals are numerous, so in the simulation results shown, only the main internal, input and output signals of each block are displayed. The results shown are for the RTL code not the synthesized netlist.

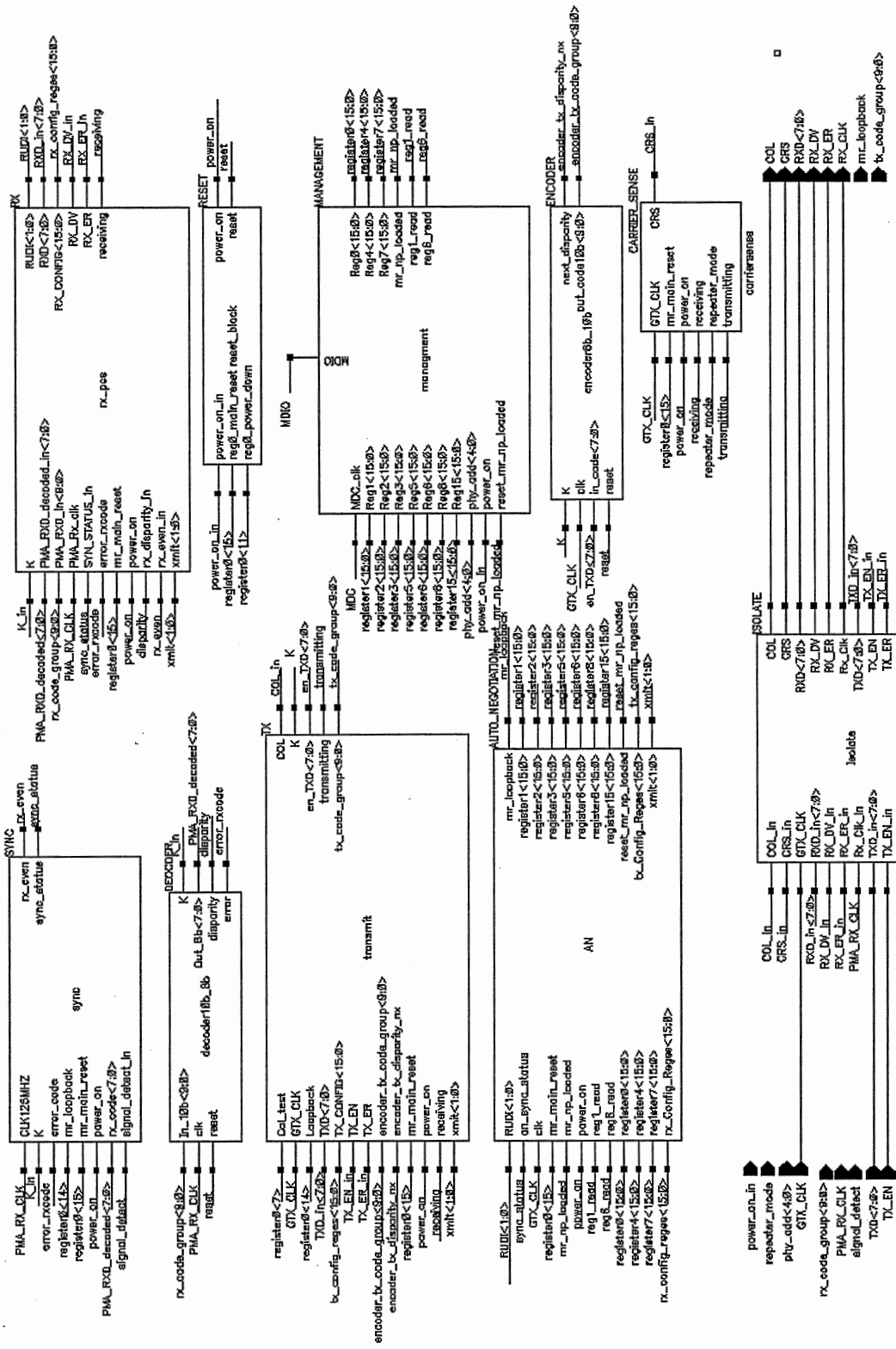


Figure 3.2 Schematic diagram for the top level PCS.

3.2.1 Transmit

As mentioned in chapter two, the transmit module is represented by two sub modules, the order-set generator and the code-group generator. Both sub modules are functionally defined by a finite state machine (FSM) [6]. Samples of the simulations are shown in figure 3.3. Only a set of signals is shown to demonstrate functionality of the modules.

The simulation shown in figure 3.3 is the start of the transmission of a packet. Signals, *TX_EN* and *TX_ER* synchronized by *GTX_CLK* are inputs from the GMII to define the packet start, end, and extension ... etc, as defined in table 2.1. For the first cycle when *TX_EN* is set to "one", *TX_ER* set to "zero", and the *TXD* data lines from the MAC layer have the preamble "55", the order-set generator will assign the *tx_o_set* to "S_order_set". The

Table 3.1 Names equivalence in schematic and hierarchy figures.

Hierarchical Name	Schematic Name
Auto Negotiation	Auto_Negotiation
Transmit	TX
Receive	RX
8B/10B encoder	ENCODER
8B/10B decoder	DECODER
Synchronization	SYNC
Carrier Sense	CARRIER_SENSE
Reset	RESET
Management	MANAGEMENT
Isolate	ISOLATE

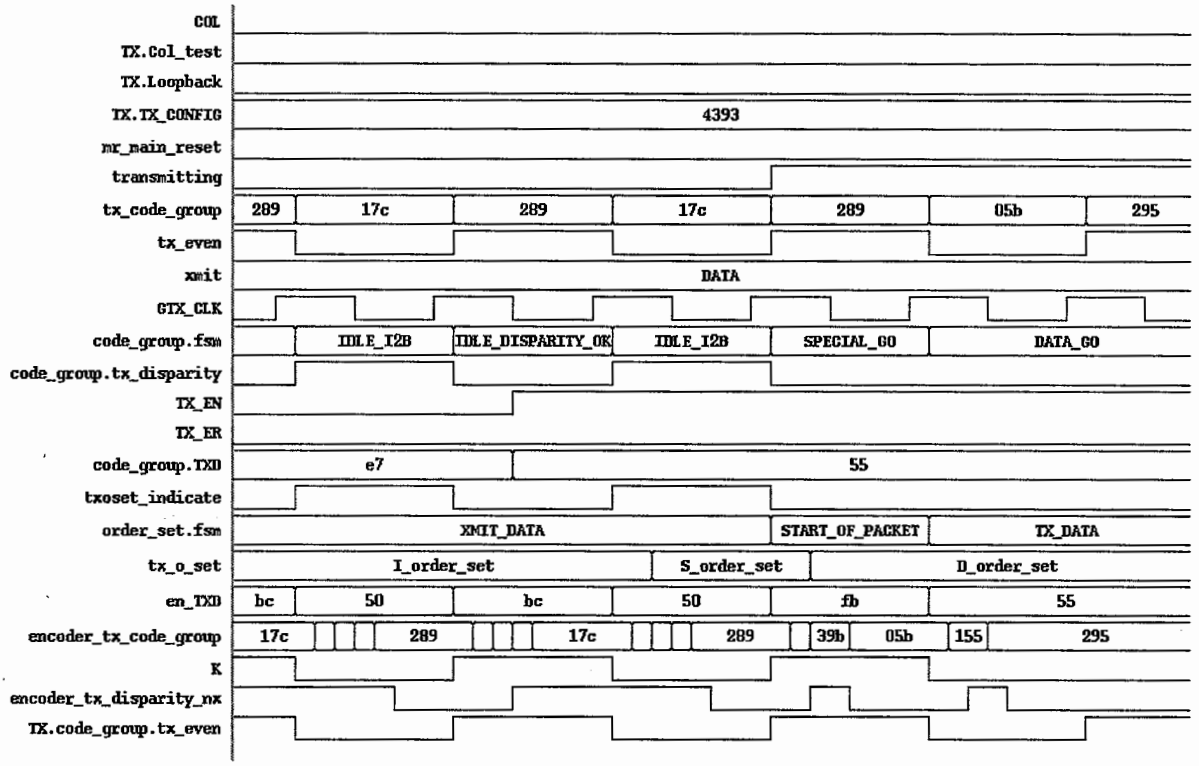


Figure 3.3 Simulation results from transmit module testing.

tx_o_set is an input to the code-group generator. The code-group generator will put the correct values of *en_TXD* and *K* to the encoder. After one cycle delay the result from the encoder will be assigned to the to the output of the transmit module to the link partner.

At the second cycle the *TX_EN* is set to one, the order-set generator assign *tx_o_set* to "D_order_set" to indicate the data of the packet. The code-group generator will assign the *en_TXD* to the input *TXD* as long as the *tx_o_set* value is "D_order_set" until the end of packet is received.

At the end of packet (it is not shown in the simulation sample), the

time the TX_EN value is "zero". The order-set generator will assign *tx_o_set* to "T_order_set" for one cycle and to "R_code_group" for one or two cycles depending on the *tx_even* value. Then the order-set generator will assign the *tx_o_set* to "I_oreder_set" to indicate idle. The transmission of the K28.5 (comma) at the beginning of the /I/ order-set, should always be done on the even code-group. If at the end of packet the last "D_code_goup" is odd an extra "R_code_group" is generated. The next comma should be generated at the even code-group, the end of packet simulation is shown in the receive process.

Both the order-set and the code-group generators have the variable *fsm* that represents the name of the state inside the FSM; Appendix B shows the state diagram for all the modules. The detailed description for the signals shown is provided at the end of this section.

3.2.2 Receive

Simulation results for the receive modules are shown in figure 3.4. This sample represents an extended packet with an error received during the extension. The extended packet shows the end of packet reception, and an extension is issued on the inputs then the start of a new packet. The input ten-bit code-groups are decoded into eight-bit code-groups using the decoder. The type of each code-group is determined to ease the decision making during reception, this is done using the *K* and the *PMA_RXD_decoded*.

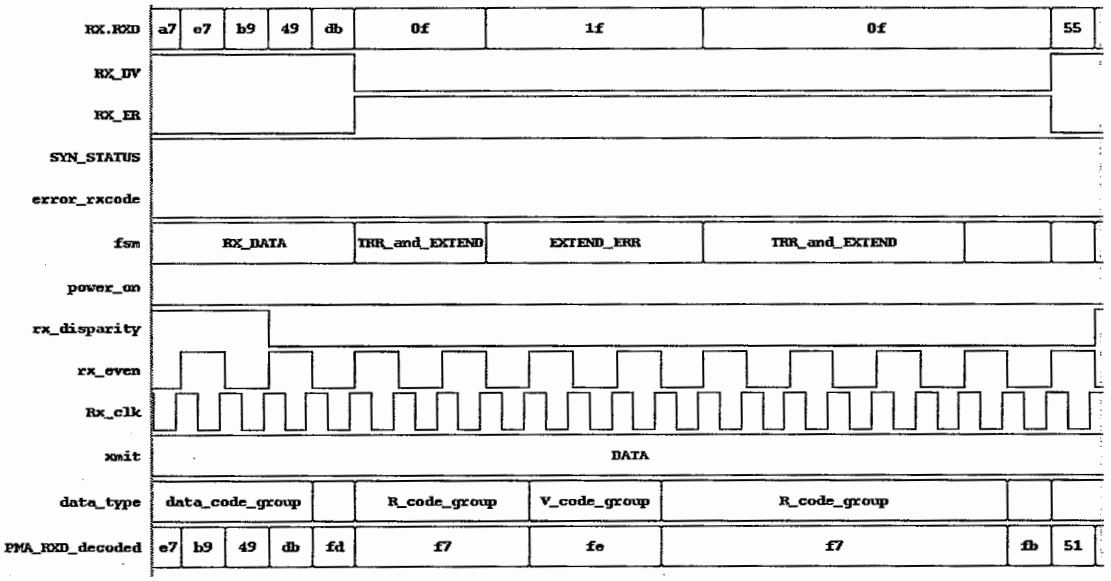


Figure 3.4 Simulation results from receive module testing.

At the end of a data packet, similar to what happen at the transmit side. End of packet is received by receiving *data_type* of "T_code_group" followed by one or more "R_code_group". During the extension the *data_type* continue to be "R_code_group". Due to an error occurred for testing purpose during the extension the *data_type* is assigned to "V_code_group". When the packet resumes the *data_type* is back to "data_code_group".

The output signals *RXD*, *RX_DV*, *RX_ER*, and *RX_CLK* go up across the GMII interface into the MAC layer. The *fsm* variable represents the state in the FSM for the receive block [6]. The detailed description for the signals shown is provided at the end of this section.

3.2.3 Auto-negotiation

The simulation and the results for this module are shown in figure 3.5. The main variable in this module is *xmit*, which tells the other modules the state of the auto-negotiation process.

The *rx_config_reg* and the *tx_config* register represents the values the configuration registers received and transmitted, respectively. The configuration register contains the basic capabilities of the PHY. The signal *link_timer_done* indicates that the *link_timer* is done, the *link_timer* times the length of time the auto-negotiation block stays in a certain state in the FSM of the auto-negotiation. Here the *link_timer* was set to a small number to decrease the simulation time. The *RUDI* is a signal from the receive module to specify the type of the code-group received. The *Link_status* from the synchronization module indicates data received is synchronized or not. The detailed description for the signals is provided at the end of this section.

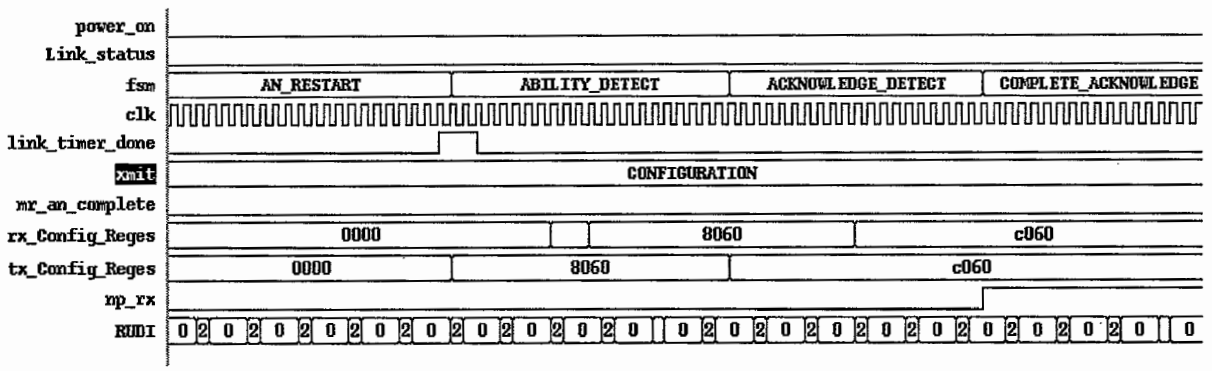


Figure 3.5 Simulation results for auto-negotiation module.

3.2.4 Synchronization

A sample for this module simulation is shown in figure 3.6. The most important signal is *sync_status*, which indicates if the synchronization has been acquired. The signal *signal_detect* comes from the PMA to indicate if a signal is received. The signal *rx_even* indicates if the received signal is odd or even. The detailed description for the signals shown is provided at the end of this section.

3.2.5 Top PCS simulation

In this section two simulation graphs are shown to illustrate a sample of the top-level simulation of the PCS.

The interface signals between the PCS and both the GMII and the PMA are shown in figure 3.7. The setup for this simulation is as follows, the *tx_code_group* signal, output of the transmitter that is usually connected into

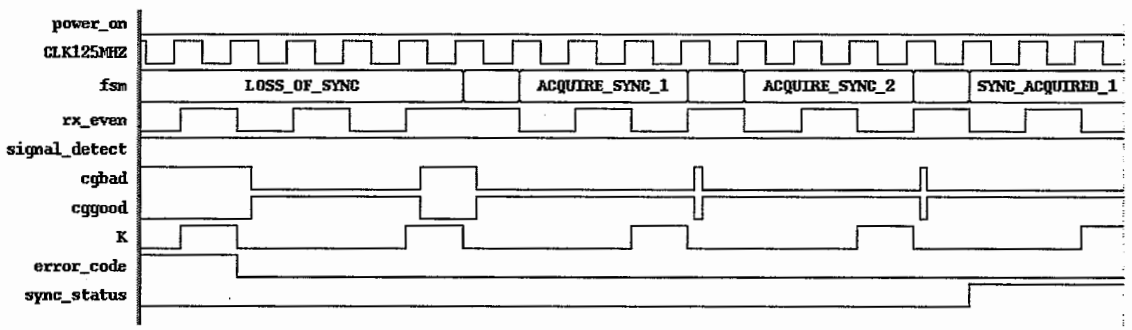


Figure 3.6 Simulatation results for synchronization module.

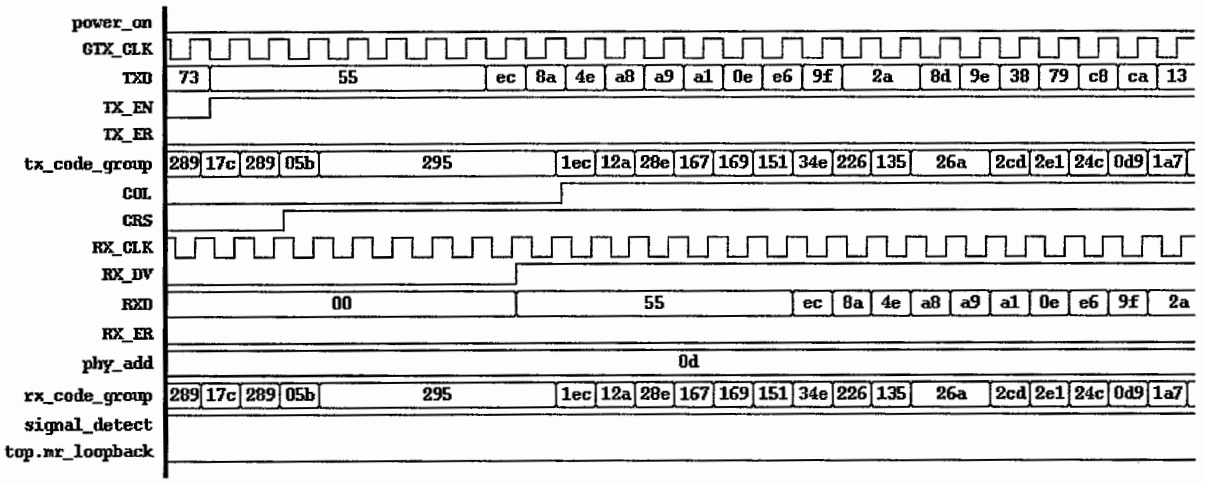


Figure 3.7 Simulations results for top level PCS.

the PMA, is connected to the *rx_code_group* signal that is input to the receiver. A loop back was constructed. This connection is different from the loopback mode defined for the physical layer since that is done with signals inside the PMA, and it is defined for the PHY as a whole. As can be seen there is a synchronous signal delay between the inputs and the outputs to the GMII, this is due to the delays along the transmission and reception paths. Comparing pairs of signals, *TXD* with *RXD*, *TX_EN* with *RX_DV*, and *TX_ER* with *RX_ER* show the delay. The same configuration was simulated with a half cycle delay between output from the transmit and input for the receive to make sure that the circuit would work when *RX_CLK* and *TX_CLK* are not synchronized.

As Part of the testing for the top level, packets from the transmit were transmitted and received by the receiver, under different conditions. Of the

conditions tested, even and odd size for the data packet, random data was transmitted, and packets with data errors, and packets with extension.

More simulation results are shown in figure 3.8. This simulation was done for the management frame transfer, where as shown in figure 3.8 the read register frame is issued. Note that for the signal *MDIO*, when it is driven neither by the PCS nor by the management entity, it is assigned to the high impedance state "high z". The signal level in figure 3.8 is shown in the middle between the logic high and logic low, since there is no color in the printout. The high impedance value is only correct for the simulation environment; the actual physical voltage level is not determined as a value. The detailed description for the signals shown is provided at the end of this section.

3.2.6 Definition for signals used in PCS

The signals used in the previous figures in section 3.1 are defined here. Notice that the variables are presented here with the same upper or lower

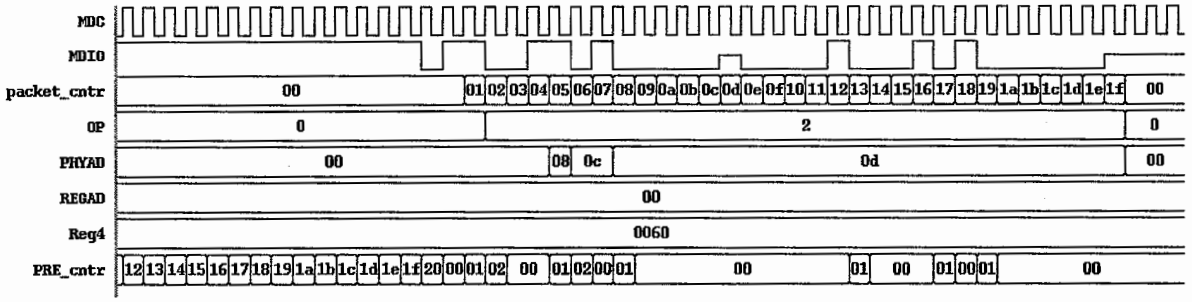


Figure 3.8 Simulation results for management top level.

case as they are used in the figures.

The input and output signals for the PCS were defined in chapter two, section 2.3, those signals at the GMII interface are *TXD*, *TX_EN*, *TX_ER*, *GTX_CLK*, *RX_DV*, *RX_ER*, *RXD*, *RX_CLK*, *COL* and *CRS*. Those signals definitions will not be repeated here.

The other signals are defined here alphabetically as follows:

- *cgbad*: this signal in the synchronization process, it indicates if the received comma is correctly aligned with respect to the order-set received. Values: "true" the signal is not aligned, "false" the signal is aligned.
- *cggood*: the inverse of cgbad.
- *CLK125MHZ*: clock signal received from the PMA, it is used in the receive and the synchronization processes.
- *data_type*: a variable that indicates the type code-group decoded, used in the receive decoding process. Values: "data_code_group" the code-group is data, "D00_code_group" the code-group is "D0.0", "D22_code_group" the code-group is "D2.2", "D56_code_group" the code-group is "D5.6", "D162_code_group" the code-group is "D16.2", "K285_code_group" the code-group is "K28.5" (comma), "R_code_group" the code-group is "K23.7", "S_code_group" the code-group is "K27.7", "T_code_group" the code-group is "K29.7", and

"V_code_group" the code-group is "K30.7".

- *error_code*: a Boolean variable to indicate an error in decoding the received code-group. Values: "true" the received code-group is not correct, "false" the received code-group is correct.
- *fsm*: this signal in all the modules to describe the FSM variable in that module, see appendix B for the state diagram for the different modules.
- *K*: a Boolean variable to indicate that the code-group is a special code-group or data code-group. Values: "true" the current code-group is special code-group, "false" the current code-group is a data code-group.
- *Link_status*: same as *SYN_STATUS*.
- *link_timer_done*: a Boolean flag that indicates when the timer *link_timer* has been done. Values: "true" the *link_timer* is done, "false" the *link_timer* is not done.
- *mr_an_complete*: a Boolean that indicate when the auto-negotiation procedure is complete. Values: "true" the auto-negotiation procedure is complete, "false" the auto-negotiation is not complete.
- *mr_loopback*: A Boolean that indicates the enabling and the disabling of the data being looped back through the PHY. Values: "true" loopback through the PHY is enabled, "false" Loopback

through the PHY is disabled..

- *mr_main_reset*: Controls the resetting of the PCS via Control Register bit 0.15. Values: "true" Reset the PCS, "false" Do not reset the PCS.
- *np_rx* : a Boolean to indicate when the next page is received. Values "true" next page is received, "false" next page is not received.
- *phy_address<4:0>*: five-bit top-level PCS input that indicate the address of the PHY in a multiple PHY system.
- *PMA_RXD_decoded<3:0>*: an eight-bit data line indicates the data decoded from the ten-bit data received from the PMA.
- *Power_on*: Condition that is true until such time as the power supply for the device that contains the PCS has reached the operating region. The condition is also true when the device has low power mode set via Control register bit 0.11. Values: "true" the device has not been completely powered, "false" the device is completely powered.
- *receiving*: A Boolean set by the PCS Receive process to indicate carrier activity. Used by the Carrier Sense process, and also interpreted by the PCS transmit process for indicating a collision. Values: "true" Carrier being received, "false" Carrier is not being

received.

- *repeater mode*: A Boolean used to make the assertion of Carrier Sense occur only in response to receive activity when the PCS is used in a CSMA/CD repeater. This variable is set to true in a repeater application, and set to false in all other applications. Values: "true" allows the assertion of CRS in response to receive activity only, "false" Allows the assertion of CRS in response to either transmit or receive activity.
- *RUDI<1:0>*: a variable set by the receive process to indicate a classification of the order-sets received. Values: "/C/" configuration order-set received, "/I/" idle order-set received, "invalid" neither configuration nor idle is received.
- *rx code-group<9:0>*: A ten-bit vector represented by the most recently received code-group from the PMA. When code-group alignment has been achieved, this vector contains precisely one code-group.
- *rx Config Reges<D15:D0>*: A 16-bit array that contains the configuration data bits received. Conveyed by the PCS Receive process to the PCS Auto-Negotiation process. The format of the data bits is context dependent, relative to the state of the Auto-Negotiation function.

- *rx_disparity*: a Boolean variable that indicate the disparity for the receiver, it is used for error detection. Values: "true" positive received disparity, "false" negative received disparity.
- *rx_even*: a Boolean variable used to align the received code-groups with respect to the comma received. Values: "true" the received code-group is aligned even, "false" the received code-group is aligned odd.
- *sync_status*: same as *SYN_STATUS*.
- *SYN_STATUS*: a Boolean variable that indicate the state of the synchronization process. Values: "true" the synchronization has been locked, "false" the synchronization is lost.
- *signal_detect*: input from the PMD to indicate there was a signal detected on the receive side. Values: "true" a signal was detected, "false" no signal was detected.
- *transmitting*: A Boolean set by the PCS transmit process to indicate carrier activity. Used by the carrier Sense process and also interpreted by the PCS transmit process for indicating a collision. Values: "true" carrier being transmitted, "false" Carrier not being transmitted.
- *tx_code_group<9:0>*: a ten-bit signal from the transmit of the PCS down to the PMA, it is the encoded data groups generated by the

code-group generator.

- *tx_Congfig_Reges<D15:D0>*: the register used to transfer information about the capabilities of the PHY to the link partner.
- *tx_disparity*: a Boolean variable that determine the disparity of the transmit process, the disparity value is used as an error detection scheme during transmission. Values: "true" positive running disparity, "false" negative running disparity.
- *tx_disparity_nx*: same definition as tx_disparity but holds the value of the tx_disparity from the current to the next clock cycle.
- *tx_even*: a Boolean variable indicates the order of the transmitted code-groups with respect to the comma sent. Values: "true", the code-group being sent is an even code-group, "false" the code-group being sent is an odd order.
- *txoset_indicate*: a Boolean variable to indicate to the order-set generator that the code-group generator has finished sending the previous order-set, since some of the order-sets has up to four code-groups. Values: "true" the order-set has been transmitted, "false" the order-set is not transmitted yet.
- *xmit*: a variable controlled by the auto-negotiation process to indicate the state of the connection. Values: "configuration" to indicate the auto-negotiation is in configuration state, "idle" to

indicate auto-negotiation is in idle, "data" to indicate the auto-negotiation is either done or not enabled.

3.3 PMA Simulation and Results

The PMA has two kinds of modules analog and digital. The analog modules are the clock_recovery and the clock_multiplier. All the other modules are digital. The schematic diagram for the PMA is shown in figure 3.9. Simulations from the analog modules are shown next. The top-level simulation from the PMA is also shown. The description for the signals is provided at the end of this section. The PMA has a much smaller number of signals compared with the PCS.

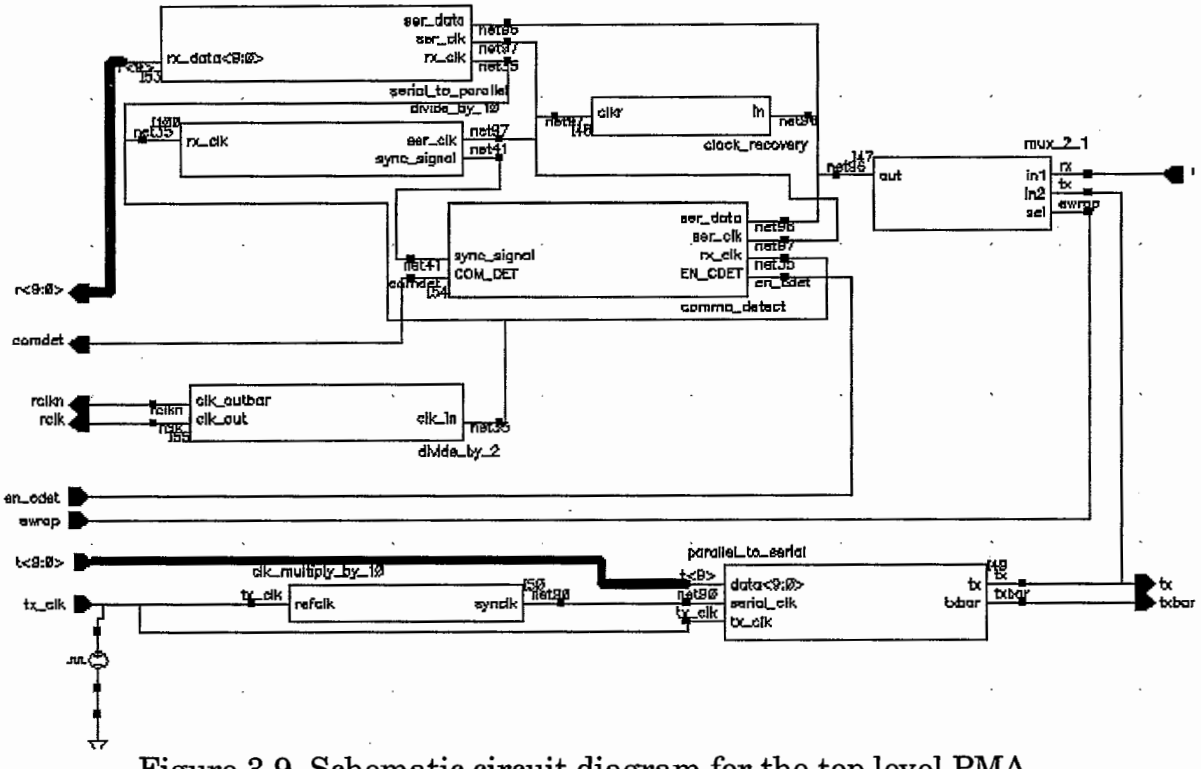


Figure 3.9 Schematic circuit diagram for the top level PMA.

As mentioned before the simulation in the PMA was done in mixed-signal environment, we have two kinds of modules in the PMA analog and digital. Through the interface elements (IE), the interface between the digital and analog modules is achieved. In the following simulations the analog parts were tested only for functional correctness, the models did not contain any analog delays, only the delays in the IE.

The analog modules were modeled using ideal equations and formulas. No parasitic or sources of error were considered.

3.3.1 Clock recovery

The schematic for the clock recovery was shown in chapter two figure 2.14 and repeated here in figure 3.10. The simulation results are shown in figure 3.11. The worst-case input is assumed; it contains the comma (K28.5),

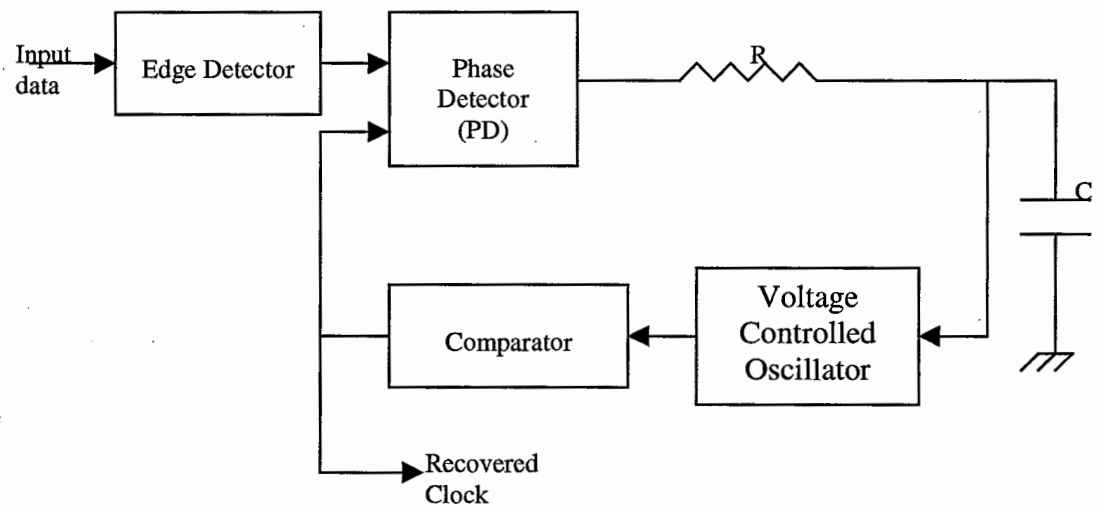


Figure 3.10 Clock cecovery schematic circuit.

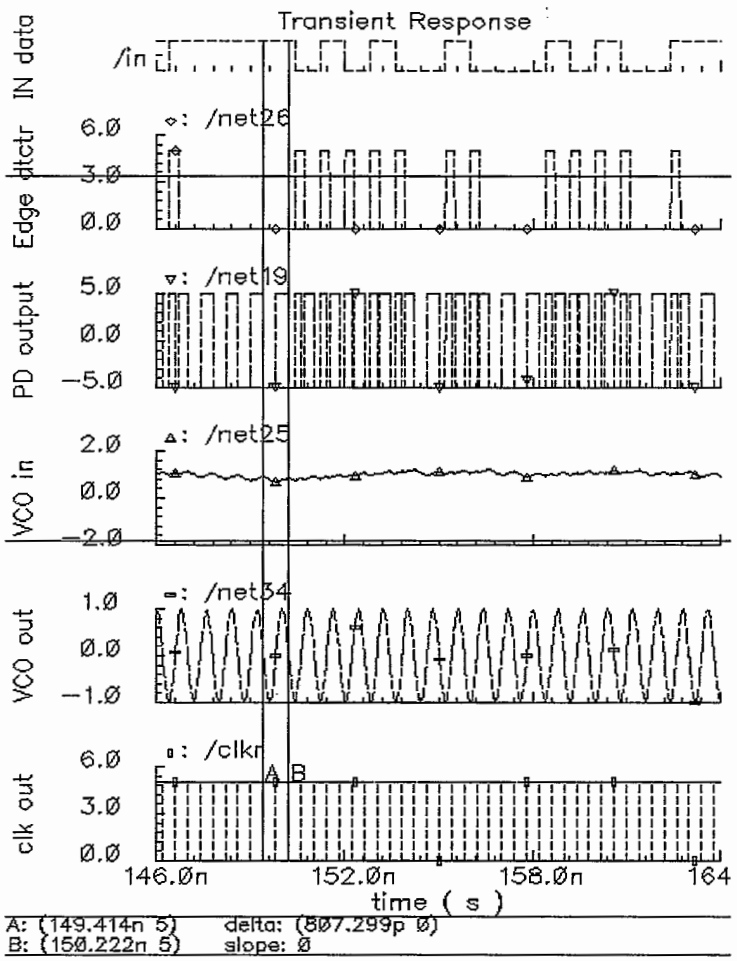


Figure 3.11 Simulation results for the clock recovery module.

which has the largest number of consecutive ones or zeros. The encoded comma with positive disparity is "110000 0101", and for the negative disparity it is "001111 1010". The comma has five consecutive same binary values. The signal names represent the signals on the schematic figure 3.11.

In the simulation results shown the serial one-bit data is applied at the input. Using the delay and an exclusive-OR gate, the edges of the input

signal are generated *Edge dtctr* (/net26) in figure 3.11. The signal is applied to the PLL to recover the clock. The voltage-controlled oscillator (VCO) was set to center frequency of 1.2GHz, compared to data frequency of 1.25MHz with a gain of 40MHz/Volt. During the time the consecutive ones are received the error is maximum. Signals at different points in the PLL are shown on figure 3.10. The signal at the output of Phase detector *PD output* (/net19). The signal at the input of the VCO *"VCO in"* (/net25). The signal at the output of the VCO *"VCO out"* (/net34)

The recovered clock clk out (/clkr) is shown also on figure 3.10. The markers A and B on the graph show the period of the recovered clock as (8.073ns), where the worst error value was (0.11ns).

3.3.2 Clock multiplier

The schematic for the clock multiplier is shown in chapter two figure 2.15 repeated here in figure 3.12. The simulation results shown in figure 3.13. Note that the input to this circuit is *GTX_CLK* clock from the PCS. The presence of a clock signal makes it easier than the clock recovery to be built. The input clock is applied to PD and with the clock generated divided by ten, the result *PD output* (/net19) is shown on figure 3.13. The input of the VCO, *VCO in* (/net25).

The output of the VCO *"VCO out"* (/net34) is the same as the clock recovery. Names of the signals are the names of the schematic figure 3.12.

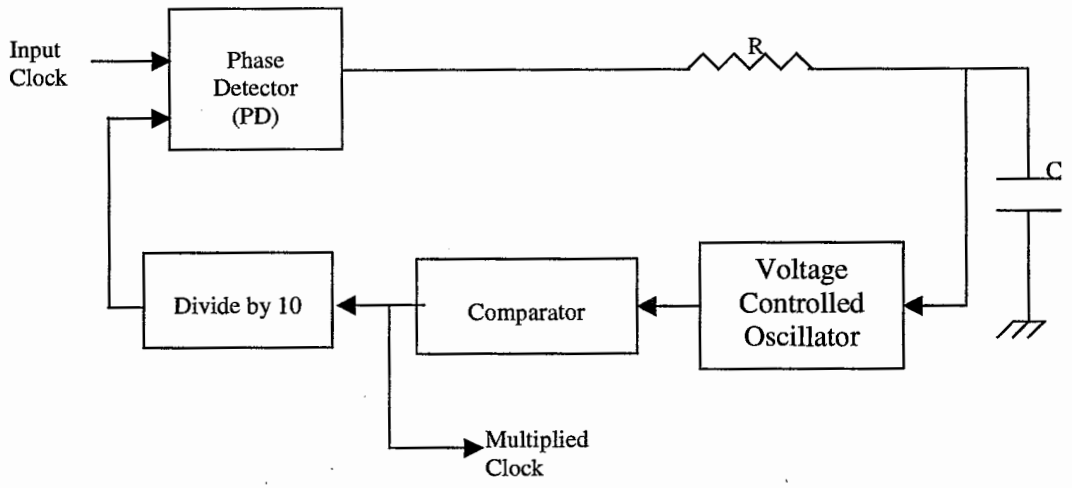


Figure 3.12 Block diagram for clock multiplier.

The markers A and B and their horizontal (x-values) under the graph, show the period at the output of the *divide_by_10* block, the period is (7.99ns). Between the markers on the resulted clock *out clk* there is 10 clock periods of (0.8ns). The error in the *out clk* is larger than that for the *Divide 10* clock, since the latter was used in the feedback loop.

3.3.3 Top level PMA

The schematic for the PMA is shown in figure 3.9. The schematic here is equivalent to the block diagram shown in figure 2.13 in chapter 2. The parallel-to-serial is for the transmission to the serial output, and the serial-to-parallel is for the reception from serial input. The functions for the clock recovery and clock multiplier were discussed in sections 2.5.1 and 2.5.2. The comma detect is to detect the comma serial bit sequence in the input, comma was mentioned in chapter two section 2.5. The divide-by-ten and the divide-

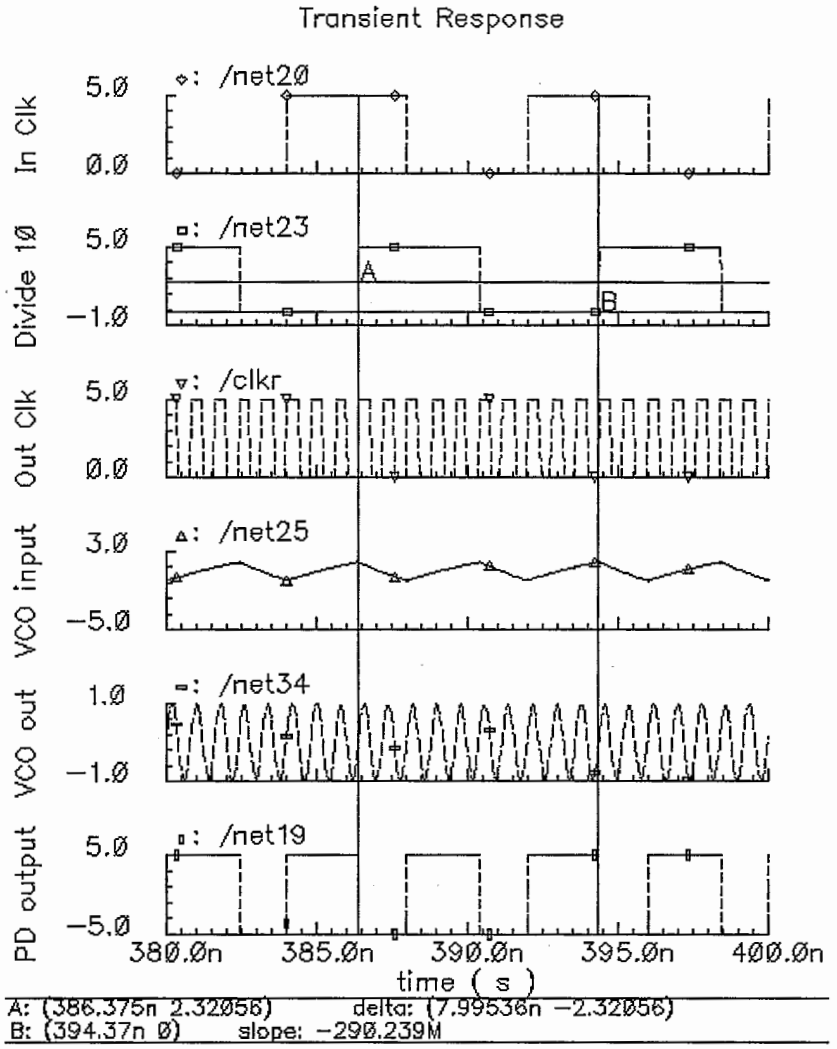


Figure 3.13 Simulation results for the clock multiplier module.

by-two are to get frequencies of 125MHz and 62.5MHz with twice the phase, respectively.

Similar to the simulation done for the PCS here we used a loopback setup for testing. Simulation results are shown in figure 3.12. Comparing *rx_code_group* (*/r<9:0>* in the figure 3.14) with *tx_code_goup* (*/t<9:0>* in figure

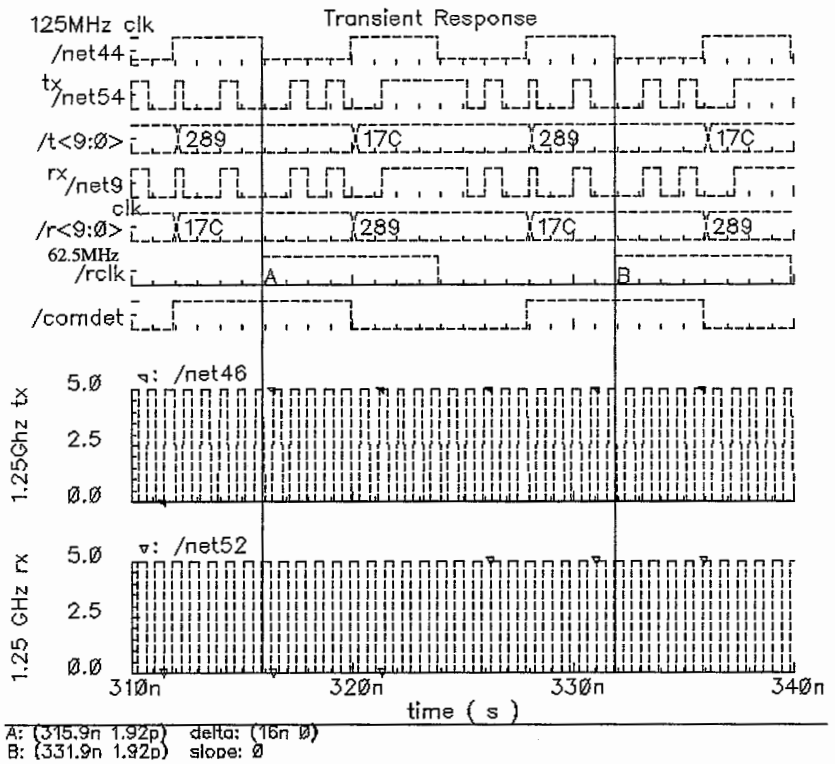


Figure 3.14 Simulation for the top level PMA.

3.14). After one clock delay, the */r<9:0>* is equal to */t<9:0>* as expected. The input contains the comma so the comma detect signal (*/comdet*) would go high for the cycle the comma is received. The markers A and B are across the *62.5MHz clk* (/rclk), the value is (16ns). The *125MHz clk* (/net44) shows two cycles between the markers. The 1.25GHz clocks *1.25GHZ tx* (/net46) and *1.25GHz rx* (/net52) show 20 cycles between the markers. The serial data is transferred from the *tx* (/net54) to the *rx* (/net9), signals are the same.

3.3.4 Definition for signals used in PMA

The signals used in the previous figures in section 3.3 are defined here.

Notice that the variables are presented here with the same upper or lower case as they are used in the PMA block diagram.

- *signal_detect*: A Boolean variable set by the PMD continuously to indicate the status of the incoming link signal. Values: "fail"; a signal is not present on the link. "OK"; a signal is present on the link.
- *tx_code-group<9:0>*: A ten-bit variable representing one code-group, which has been prepared for transmission by the PCS transmit process.
- *PMA_TX_CLK*: The 125 MHz transmit code-group clock. This code-group clock is used to latch data into the PMA for transmission. The transmitter clock multiplier unit to generate the 1,250 MHz bit rate clock also uses *PMA_TX_CLK*.
- *EWRAP*: enables the PMA to electrically loop transmit data to the receiver. The serial outputs on the transmitter are held in a static state during *EWRAP* operation. Equivalent to the *loopback* variable.
- *rx_code-group<9:0>*: Presents the ten-bit parallel receive code-group data to the PCS. When code-groups are properly aligned, any received code-group containing a comma is clocked by *PMA_RX_CLK<1:0>*.

- *PMA_RX_CLK<0>*: The 62.5 MHz receive clock that the protocol device uses to latch odd-numbered code-groups in the received PHY bit stream. This clock may be stretched during code-group alignment, and is not shortened.
- *PMA_RX_CLK<1>*: The 62.5 MHz receive clock that the protocol device uses to latch even-numbered code-groups in the received PHY bit stream. *PMA_RX_CLK<1>* is 180° out-of-phase with *PMA_RX_CLK<0>*.
- *COM_DET*: An indication that the code-group associated with the current *PMA_RX_CLK<1>* contains a valid comma
- *LCK_REF*: Causes the TBI clock recovery unit to lock to PMA_TX_CLK.
- *EN_CDET*: Enables the PMA to perform the code-group alignment function on a comma. When *EN_CDET* is asserted the code-group alignment function is operational. The PMA client optionally generates this signal. The PMA sublayer may leave this function always enabled.

3.4 Chapter Conclusion

In this chapter results for the PHY simulations were presented, and a description of the internal signals was also provided. The simulations were verified by configuring the PHY in the loopback mode, where it can

communicate with itself as if it is communicating with a link partner.

During code simulation a delay estimate was assumed to help in giving more practical results. A delay of "0.5ns" was added to every combinational code assignment and a delay of "1ns" for each register (flip-flop) assignment.

4. PCS SYNTHESIS

4.1 Introduction

In the previous chapter we showed the simulations for the PCS. In this chapter the PCS synthesis will be discussed.

Synthesis is done using the Synopsys® Design Compiler® tool. The synthesis goal is to generate a standard cell netlist that fulfils the design and the cell library constraints. A standard cell is a pre-made cell (layout is ready), in such a way that the inputs, outputs and power lines are configured in order to enable the tailoring of a large number of the standard cells into a design. The standard cells are defined through a technology library that is related to a specific technology. The technology library contains all the characteristics about all the cells defined, this includes the logical function, timing delays between all the inputs and all the outputs, the loading effect, the driving power, ...etc. A commercial 0.35μ technology library was used in the synthesis.

In addition to the technology library and the constraints, the synthesis tool requires the definition of the clocks.

The input and the output variables constraints are defined, these include timing, delay, and loading. The reports generated by the synthesis tool are also presented.

4.2 Input and Output Signals Characteristics

We have inputs and outputs to the PCS at the top layer of the GMII shown and at the lower layer of the PMA. See figure 2.4 for the GMII interface signals, and figure 2.13 for the PMA interface signals. In the synthesis process the characteristics of the inputs, outputs, and the clocks should be determined. Following is the discussion about the characteristics of those signals.

4.2.1 GMII interface inputs and outputs

The set of timing values needed for synthesis of inputs, outputs, and clocks signals at the GMII interface to the PCS are shown in table 4.1 [6]. The table specifies the AC characteristics for the GMII interface signals. It gives the Min and Max values where applicable. Note that some of these signals are input and others are output across the GMII interface.

For the clocks GTX_CLK and RX_CLK we have the following parameters. Acceptable period (t_{PERIOD}), the acceptable value for the period the clock, there is maximum and minimum values for GTX_CLK and minimum for RX_CLK. Time low (t_{LOW}) and time high (t_{HIGH}), the time the clock stays low or high respectively, as part of the period, for both clocks we have a minimum. Rise time (t_R) and fall time (t_F); the signal transition time needed to go from low to high and from high to low, respectively, there is a maximum for both clocks. Definitions are shown graphically in figures 4.1 and 4.2.

Table 4.1 AC specifications.
(From IEEE Std. 802.3z-1998,©1998 All rights reserved [6])

Symbol	Parameter	Conditions	Min	Max	Units
V_{IL_AC}	Input Low Voltage AC	–	–	0.70	V
V_{IH_AC}	Input High Voltage AC	–	1.90	–	V
f_{FREQ} GTX_CLK	Frequency	–	125 – 100ppm	125 + 100ppm	MHz
t_{PERIOD} GTX_CLK	Period	–	7.50	8.50	ns
t_{PERIOD} RX_CLK	Period	–	7.50	–	ns
t_{HIGH} GTX_CLK, RX_CLK	Time High	–	2.50	–	ns
t_{LOW} GTX_CLK, RX_CLK	Time Low	–	2.50	–	ns
t_R GTX_CLK, RX_CLK	Rise Time	$V_{IL\_AC(max)}$ to $V_{IH\_AC(min)}$	–	1.00	ns
t_F GTX_CLK, RX_CLK	Fall Time	$V_{IH\_AC(min)}$ to $V_{IL\_AC(max)}$	–	1.00	ns
–	Magnitude of GTX_CLK, RX_CLK Slew Rate (rising)[a]	$V_{IL\_AC(max)}$ to $V_{IH\_AC(min)}$	0.6	–	V/ns
–	Magnitude of GTX_CLK RX_CLK Slew Rate (falling)[a]	$V_{IH\_AC(min)}$ to $V_{IL\_AC(max)}$	0.6	–	V/ns
t_{SETUP}	TXD, TX_EN, TX_ER Setup to •GTX_CLK and RXD, RX_DV, RX_ER Setup to •RX_CLK	–	2.50	–	ns
t_{HOLD}	TXD, TX_EN, TX_ER Hold from •GTX_CLK and RXD, RX_DV RX_ER Hold from, •RX_CLK	–	0.50	–	ns
$t_{SETUP(RCVR)}$	RX_ER Setup to •RX_CLK	–	2.00	–	ns
$t_{HOLD(RCVR)}$	TXD, TX_EN, TX_ER Hold from •GTX_CLK and RXD, RX_DV, RX_ER Hold from •RX_CLK	–	0.00	–	ns

a. Clock Skew rate is the instantaneous rate of change of the clock potential with respect to time (dV/dt), not an average value over the entire rise or fall time interval. Conformance with this specification guarantees that the clock signals will rise and fall monotonically through the switching region.

For the input and output signals the values for the setup time (t_{SETUP}), the time the signal should be ready before the edge of the clock, there is a minimum for all the signals. The hold time (t_{HOLD}), the time the signal should hold its value after the clock edge, there is a minimum value for all the

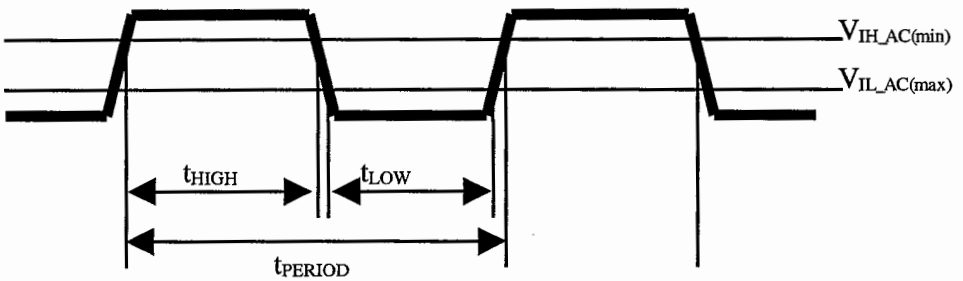


Figure 4.1 GTX_CLK and RX_CLK timing parameters at receiver input. (From IEEE Std. 802.3z-1998,©1998 All rights reserved [6]).

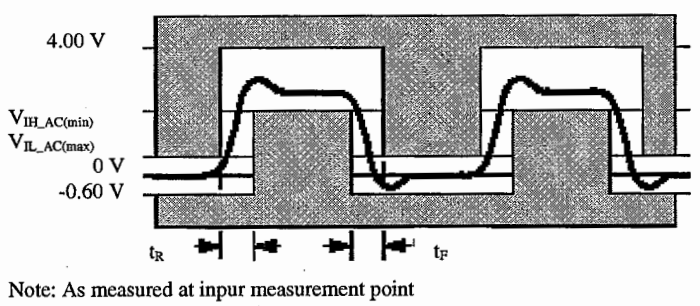


Figure 4.2 GMII receiver input rise and fall time definitions. (From IEEE Std. 802.3z-1998,©1998 All rights reserved [6])

signals. The minimum values are given in table 4.1. The definitions of t_{HOLD} and t_{SETUP} are shown graphically in figure 4.3. Note in figure 4.3 that the clock is either of GTX_CLK or RX_CLK.

Another sets of values of the inputs and outputs are needed for the synthesis tool, those are listed next. The maximum delay time with respect to the clocks t_{DELAY}, defined to be 1ns. The maximum driving capabilities for the inputs and the outputs, max drive is set to 1, this indicates the output

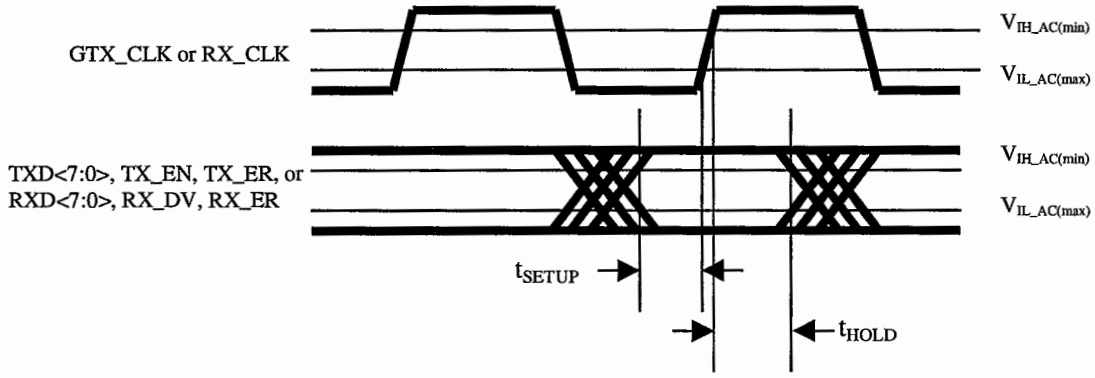


Figure 4.3 GMII signal timing at receiver input.
(From IEEE Std. 802.3z-1998,©1998 All rights reserved [6])

resistance in KΩ of the driving cell. The maximum capacitive load at the outputs, defined to be 5pF.

4.2.2 PMA interface inputs and outputs

Similar to the definitions for the GMII signals, the requirements for the PMA signals are shown in tables 4.2 and 4.3. The definitions for hold time, setup time, rise time, fall time are similar to definitions in the previous section about the GMII signals.

4.2.3 Management interface inputs and outputs

The management signals are the MDIO inout and the MDC clock. The setup and hold requirement for the MDIO are both 10ns [14]. The minimum period for the MDC clock is 400ns with minimum t_{LOW} and t_{HIGH} times of 160ns [14]. The uncertainty in the high and low times results in more effort needed to perform the synthesis for worst-case clock.

Table 4.2 DC specifications for the PMA interface signals.
(From IEEE Std. 802.3z-1998,©1998 All rights reserved [6]).

Symbol	Parameter	Conditions		Min	Type	Max	Units
VOH	Output High Voltage	I OH =-400uA	Vcc =Min	2.2	3.0	V CC	V
VOL	Output Low Voltage	I OL =1mA	V CC =Min	GND	0.25	0.6	V
VIH	Input High Voltage			2.0	-	V CC [a] +10%	V
VIL	Input Low Voltage			GND	-	0.8	V
IIH	Input High Current	V CC =Max	V IN =2.4V	-	-	40	•A
IIL	Input Low Current	V CC =Max	V IN =0.4V	-	-	600	•A
CIN	Input Capacitance			-	-	4.0	pf
tR	Clock Rise Time	0.8V to 2.0V		0.7	-	2.4	ns
tF	Clock Fall Time	2.0V to 0.8V		0.7	-	2.4	ns
tR	Data Rise Time	0.8V to 2.0V		0.7	-	-	ns
tF	Data Fall Time	2.0V to 0.8V		0.7	-	-	ns

a. Refers to the driving device power supply.

4.2.4 Other interface inputs and outputs

The other input and output signals are not synchronized with a clock, some of those signals are constant such as the *phy_address* and some of them change without being related to the clocks, such as the *power_on* signal. If a change occurs on any of those signals, the change would not be synchronized and the effect is also not synchronized. So the nets connected to this input are not optimized during synthesis, since it is not known when the change arrives any way. Also those signals are changing across multiple clock cycles.

4.3 Synthesis Reports

The synthesis was done using constraints for inputs and outputs as defined in the previous section. In the definition of multi-clock system for synthesis, the relation between the clocks should be stated, but since the system is partially divided between transmit and receive sides each with its clock domain. The interaction between signals from one clock domain to the other does not need to be synchronous. The clocks were defined to be one clock to make it easier for the synthesis process. For example the *GTX_CLK* and the *RX_CLK* are related by the auto-negotiation process, where the receive process sends the received configuration register to the auto-negotiation which works in the *GTX_CLK* domain, and for that an extra or less one cycle will not be different. In reality the clock delay relation is random,local clock generates *GTX_CLK* while input data generates RX_CLK.

The technology library used for the synthesis is a 0.35μ CMOS library. The design was synthesized as one unit, which means all the modules were ungrouped into one module. According to Synopsys® [16] it is recommended if the design is less than 5000 gates. In addition, the ungrouping was done with the flatten option, which will continue the ungrouping process across the hierarchy until the gate level is reached. This procedure will take more time but it will provide better synthesis results.

Area is an important constraint that we need to minimize, the area

constraint was set to zero to achieve maximum minimization from the tool. There was no power constraint. Reports for the synthesis are shown in figures 4.4 - 4.6.

In the area report shown in figure 4.4, the total cell area is mentioned, also the combinational logic area and noncombiantional (sequential) logic area is also shown. The report shows the number of cells used. Also the area report shows the number of ports, nets, cells and references (cells types) used. The area of the design is considered to medium.

```
****************************************
Report : area
Design : pcs_top
Version: 1999.05-2
Date   : Sat Nov  6 20:21:54 1999
****************************************

Library(s) Used:

   fs80a_a (File: /home/process/UMC/UM35/synopsys/faraday/fs80a_core.db)

Number of ports:           56
Number of nets:          4790
Number of cells:         4295
Number of references:      91

Combinational area:     13702.000000
Noncombinational area:   8783.000000
Net Interconnect area:     undefined  (Wire load has zero net area)

Total cell area:        22485.000000
Total area:              undefined
1
Note: The total area is not shown since this includes the area for the wiring between the cells, which is not known
before the placement and routing of the cells.
```

Figure 4.4 Area report for the synthesis of the PCS.

```
****************************************
Report : constraint
Design : pcs_top
Version: 1999.05-2
Date   : Sat Nov  6 20:21:56 1999
****************************************


                                  Weighted
Group (max_delay/setup)    Cost    Weight    Cost
--------------------------------------------------
GTX_CLK                    0.00    1.00      0.00
MDC                        0.00    1.00      0.00
PMA_RX_CLK                 0.00    1.00      0.00
default                    0.00    1.00      0.00
--------------------------------------------------
max_delay/setup                              0.00

                        Total Neg  Critical
Group (critical_range)  Slack      Endpoints  Cost
--------------------------------------------------
GTX_CLK                 0.00       0          0.00
MDC                     0.00       0          0.00
PMA_RX_CLK              0.00       0          0.00
default                 0.00       0          0.00
--------------------------------------------------
critical_range                     0.00

                                  Weighted
Group (min_delay/hold)     Cost    Weight    Cost
--------------------------------------------------
GTX_CLK                    0.00    1.00      0.00
MDC                        0.00    1.00      0.00
PMA_RX_CLK                 0.00    1.00      0.00
default                    0.00    1.00      0.00
--------------------------------------------------
min_delay/hold                               0.00


Constraint                          Cost
--------------------------------------------------
multiport_net                       0.00 (MET)
max_transition                      0.00 (MET)
max_fanout                          0.00 (MET)
max_capacitance                     0.00 (MET)
max_delay/setup                     0.00 (MET)
critical_range                      0.00 (MET)
min_delay/hold                      0.00 (MET)
max_area                            22485.00 (VIOLATED)


1
```

Figure 4.5 Constraints report for the synthesis of the PCS.

Figure 4.5 shows the constraints report for the synthesis of the PCS. The first three parts in the report shows the details of the results for the optimization to clocks GTX_CLK, RX_CLK, and MDC domains, for the constraints setup time, critical-range, and hold time. The optimization for multiple constraints is done through a cost function, where each constraint has a weight, and the goal of the optimization is to minimize the cost function. The last part in the report states the result of the optimization for the different constraints, those are the maximum transition, maximum fanout, maximum capacitance, setup time, hold time, and maximum area. All the constraints are met except the area which was violated. As has been mentioned before it was set to zero.

The last report shown in figure 4.6 is for the timing report, where it shows the worst timing for the signal path from one register to another register through combinational logic (critical path). The critical path is shown for all the clock signals domains, they are *GTX_CLK*, *RX_CLK*, and *MDC*.

An important value in these reports is the slack time; it shows the delay time the signals have with respect to the required time. When the slack is positive, it means there is extra time for the signal to arrive before it is required. When the slack is negative, it means the signal arrives later than the required time.

Note that in the timing report the worst slack for the clocks *GTX_CLK* and *RX_CLK* domains is zero. This occurs due to the way the optimization occurs and its heuristic nature. First, the tool will optimize the circuits until the timing requirements are met, during the first pass of optimization the area is not optimized. When the timing constraints are just met, the first pass finish and the second starts. During the second pass the area is optimized while verifying the other constraints, the tool will continue until it cannot optimize the circuit area anymore. Since the area goal was set to zero, the area optimization will stop when any change to the area toward a smaller value, will violate the timing constraints.

For the *MDC* domain, since that clock is too slow for this technology the result is the very long slack time.

The timing report for the *GTX_CLK* clock is shown in figure 4.6(a). Here the worst (one or more critical path) for the *GTX_CLK* domain is shown. In the report there are two groups of signal delays used to compute the slack of a signal.

The first is the delays in the signal path starting from an input or a register and ending in an output or a register. The signal goes through all the combinational gates, where the gates delay is shown explicitly for every gate the signal went through

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : pcs_top
Version: 1999.05-2
Date   : Sat Nov  6 20:21:56 1999
****************************************

Operating Conditions: WCCOM   Library: fs80a_a
Wire Load Model Mode: enclosed

  Startpoint: RX/RX_CONFIG_reg[1]
              (rising edge-triggered flip-flop clocked by PMA_RX_CLK)
  Endpoint: AUTO_NEGOTIATION/link_timer_reg[0]
              (rising edge-triggered flip-flop clocked by GTX_CLK)
  Path Group: GTX_CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  pcs_top            G10K                  fs80a_a

  Point                                    Incr       Path
  ------------------------------------------------------------------------------
  clock PMA_RX_CLK (rise edge)                        0.00       0.00
  clock network delay (ideal)                         1.00       1.00
  RX/RX_CONFIG_reg[1]/CK (DFCLRBN)                    0.00       1.00 r
  RX/RX_CONFIG_reg[1]/Q (DFCLRBN)                     0.83       1.83 f
  U2319/O (NR2P)                                      0.20       2.03 r
  U40/O (ND2P)                                        0.23       2.26 f
  U1323/O (NR2T)                                      0.20       2.47 r
  U1322/O (ND2T)                                      0.23       2.70 f
  U1967/O (INV3)                                      0.14       2.84 r
  U1963/O (ND2T)                                      0.20       3.04 f
  U1794/O (INV2)                                      0.11       3.15 r
  U1880/O (OAI12)                                     0.21       3.36 f
  U1881/O (ND2)                                       0.12       3.48 r
  U1882/O (OAI12)                                     0.19       3.67 f
  U1883/O (ND2)                                       0.19       3.86 r
  U1884/O (ND3)                                       0.40       4.25 f
  U1948/O (ND2P)                                      0.24       4.49 r
  U1338/O (INV2)                                      0.17       4.66 f
  U18/O (INV3)                                        0.15       4.81 r
  U1789/O (ND3)                                       0.61       5.42 f
  U1790/O (INV2)                                      0.14       5.56 r
  U275/O (OA12P)                                      0.45       6.01 r
  U2369/O (INV3)                                      0.47       6.48 f
  U1428/O (NR2P)                                      0.22       6.70 r
  AUTO_NEGOTIATION/link_timer_reg[0]/D (DFCLRBN)      0.00       6.70 r
  data arrival time                                              6.70

  clock GTX_CLK (rise edge)                           8.00       8.00
  clock network delay (ideal)                         1.00       9.00
  clock uncertainty                                  -1.00       8.00
  AUTO_NEGOTIATION/link_timer_reg[0]/CK (DFCLRBN)     0.00       8.00 r
  library setup time                                 -1.30       6.70
  data required time                                             6.70
  ------------------------------------------------------------------------------
  data required time                                             6.70
  data arrival time                                             -6.70
  ------------------------------------------------------------------------------
  slack (MET)                                                    0.00
```

Figure 4.6 Timing reports for the synthesis of the PCS
(a) GTX_CLK clock domain.

The second group of data represents the delays in the clock path, clock delays, clock uncertainty and the library setup requirements. For the design to work properly the signal delay through the logic should be less than or equal to the clock delay.

The timing report for MDC clock is shown in figure 4.6(b). The same argument about the report applies here, but since the clock is slow relative to the technology we have a large positive slack time.

The timing report for the RX_CLK clock is shown in figure 4.6(c). This is also similar to the previous timing reports.

Similar to the synthesis flow diagram in figure 1.7, many iterations were made through the RTL code until the optimization was done correctly. Since the violations were in the timing constraints, introducing an extra cycle into the combinational logic path to shorten the delays achieved the wanted results. The tool made this iteration easier through the use of script files to handle repeated actions, once it is defined the scripts are used to perform all the steps of optimization. The scripts used are in appendix C.

The synthesis tool generated a gate level netlist in Verilog® HDL, also it generated a standard delay format file (SDF) that describes the timing delays of the gate netlist, which describes the delay across the gates. The SDF file is generated from the timing information of the technology library used for synthesis.

```
Startpoint: MANAGEMENT/Reg0_reg[15]
          (rising edge-triggered flip-flop clocked by MDC)
 Endpoint: MANAGEMENT/reg_read_reg[2]
          (rising edge-triggered flip-flop clocked by MDC)
 Path Group: MDC
 Path Type: max

 Des/Clust/Port     Wire Load Model      Library
 ------------------------------------------------
 pcs_top            G10K                 fs80a_a

 Point                                    Incr       Path
 ------------------------------------------------------------------------------
 clock MDC (rise edge)                    0.00       0.00
 clock network delay (ideal)              1.00       1.00
 MANAGEMENT/Reg0_reg[15]/CK (DFCLRBN)                0.00       1.00 r
 MANAGEMENT/Reg0_reg[15]/Q (DFCLRBN)                 0.65       1.65 r
 U2258/O (OR3)                            0.34       1.99 r
 U4455/O (INV1)                           0.14       2.12 f
 U4456/O (INV1)                           0.28       2.41 r
 U1126/O (INV1)                           0.24       2.64 f
 U4445/O (BUF1)                           0.44       3.08 f
 U2383/O (INV3)                           0.72       3.80 r
 U1265/O (INV2)                           1.28       5.08 f
 U1131/O (ND2)                            0.57       5.65 r
 U1132/O (INV1)                           0.37       6.02 f
 U1173/O (AN2)                            0.82       6.84 f
 U2246/O (OR4B3L)                         0.41       7 25 r
 U2600/O (DEL10)                          15.10      22.35 r
 U2601/O (INV1L)                          0.46       22.81 f
 U1730/O (MAOI1)                          0.33       23.13 r
 U1462/O (ND4)                            0.33       23.47 f
 U2671/O (DEL10)                          16.27      39.74 f
 MANAGEMENT/reg_read_reg[2]/D (DFCLRBN)              0.00       39.74 f
 data arrival time                                   39.74

 clock MDC (rise edge)                    400.00     400.00
 clock network delay (ideal)              1.00       401.00
 clock uncertainty                        -1.00      400.00
 MANAGEMENT/reg_read_reg[2]/CK (DFCLRBN)             0.00       400.00 r
 library setup time                       -1.30      398.70
 data required time                                  398.70
 ------------------------------------------------------------------------------
 data required time                                  398.70
 data arrival time                                   -39.74
 ------------------------------------------------------------------------------
 slack (MET)                                         358.96
```

Figure 4.6 (Continued)
(b) MDC clock domain.

```
Startpoint: RX/data_type_reg[0]
          (rising edge-triggered flip-flop clocked by PMA_RX_CLK)
 Endpoint: RX/RX_CONFIG_reg[0]
          (rising edge-triggered flip-flop clocked by PMA_RX_CLK)
 Path Group: PMA_RX_CLK
 Path Type: max

 Des/Clust/Port     Wire Load Model      Library
 ------------------------------------------------
 pcs_top            G10K                 fs80a_a

 Point                                Incr       Path
 ------------------------------------------------------------
 clock PMA_RX_CLK (rise edge)                0.00       0.00
 clock network delay (ideal)                 1.00       1.00
 RX/data_type_reg[0]/CK (DFFN)               0.00       1.00 r
 RX/data_type_reg[0]/Q (DFFN)                0.95       1.95 f
 U1919/OB (MXL2)                             0.43       2.38 r
 U4558/O (OAI112L)                           0.52       2.90 f
 U174/O (AN2)                                0.40       3.30 f
 U1803/O (OAI12)                             0.26       3.56 r
 U1546/O (ND3)                               0.42       3.98 f
 U1237/O (NR2L)                              0.33       4.30 r
 U1215/O (MAOI1P)                            0.63       4.94 r
 U1268/O (ND2T)                              0.28       5.21 f
 U1836/O (INV2)                              0.12       5.33 r
 U69/O (ND2)                                 0.38       5.71 f
 U1218/O (OA12)                              0.47       6.18 f
U1051/O (INV2)                               0.27       6.45 r
 RX/RX_CONFIG_reg[0]/LD (DFCLRBN)            0.02       6.47 r
 data arrival time                                      6.47

 clock PMA_RX_CLK (rise edge)                8.00       8.00
 clock network delay (ideal)                 1.00       9.00
 clock uncertainty                          -1.00       8.00
 RX/RX_CONFIG_reg[0]/CK (DFCLRBN)            0.00       8.00 r
 library setup time                         -1.53       6.47
 data required time                                     6.47
 ------------------------------------------------------------
 data required time                                     6.47
 data arrival time                                     -6.47
 ------------------------------------------------------------
 slack (MET)                                            0.00


1
```

Figure 4.6 (Continued)
(c) RX_CLK clock domain.

The Verilog® HDL code and the SDF file for the PCS are simulated using the Verilog® simulator. The simulation tests used are the same as the tests for the RTL code, the results verified that the synthesis results are correct.

4.4 Chapter Conclusion

In this chapter the synthesis for the PCS was discussed. The input and output signals specifications were defined. The PCS was synthesized correctly and the synthesis reports included in the chapter showed correctness of the synthesis, also the resulting gate level netlist with the timing information were simulated in a similar way to the original code with the inclusion of the actual delay and loading effects of the standard cells used, the results are correct compared to pre-synthesis, which further shows the correctness of the synthesis results.

5. CONCLUSIONS

5.1 Discussion

The standards represent a process by which an entire industry can cooperate among its members to define new technologies that will benefit the participants and ultimately the consumer. Ethernet is the backbone for wired networking; it continues to increase in its distribution and use. The previous Ethernet standard was at a speed of 100MB/s which was available to desktop applications; however, with the increase in the size of the Internet, it was apparent that the standards needed to move fast to avoid a bandwidth problem in the near future. The normal evolution was to increase the speed of the same available techniques and make them backward compatible rather than starting a new product to get to that speed.

There are many applications today that are involved with communications, starting with the Internet, through HDTV, to cellular phones. All these are mixed-signal (analog and digital) systems. It is imperative to be able to characterize both kinds of circuits and their interactions, at least at the functional level.

5.2 Conclusion

In this work the mixed-signal system of the latest fiber IEEE Ethernet standard, the Gigabit Ethernet 802.3z standard (1000BASE-X PHY), was functionally verified, and the PHY's two sublayers, the PMA and the PCS,

were modeled.

The PCS, which is totally digital, was modeled using Verilog®HDL at the RTL level, and functionally verified through simulation using Verilog-XL®. In addition, the PCS was synthesized successfully using the Synopsys Design Compiler® tool into a gate level 0.35μ CMOS technology dependent netlist. The resulting netlist has commercial value in today's market due to the proven worthiness of the Gigabit Ethernet.

The synthesized PCS was verified through the simulation of the standard delay format (SDF) information generated by the synthesis tool from the technology library used.

The PMA is a mixed-signal system. It was modeled by SpectreHDL® for the analog parts and by Verilog®HDL for the digital parts. The PMA was functionally verified using the mixed-signal simulator SpectreVerilog®.

This work has a commercial value and was done in accordance with a company's specifications. The novelty of this work lies in applying the mixed-signal simulation tools to such a complex networking system as the IEEE 802.3z 1000BASE-X Standard (Gigabit Ethernet). This work has an importance in the challenging process of designing mixed-signal systems.

Finally this work was presented in a conference paper [17].

APPENDIX A

DATA CODE-GROUPS

In this appendix the valid data code-groups are shown, the naming convention for the code-group name is as follows.

Divide the eight-bit octet into three-bit and five-bit fields as shown in column three in table A.1, D letter in the name denote (Data) versus K for special code groups. The first number represent the decimal value of the five-bit field, and the second number denote the value of the three-bit field. This table is from [6].

Table A.1(a) Valid data code-groups

Code Group Name	Octet Value	Octet Bits HGF EDCBA	Current RD	Current RD +
			abcdei fghj	abcdei fghj
D0.0	00	000 00000	100111 0100	011000 1011
D1.0	01	000 00001	011101 0100	100010 1011
D2.0	02	000 00010	101101 0100	010010 1011
D3.0	03	000 00011	110001 1011	110001 0100
D4.0	04	000 00100	110101 0100	001010 1011
D5.0	05	000 00101	101001 1011	101001 0100
D6.0	06	000 00110	011001 1011	011001 0100
D7.0	07	000 00111	111000 1011	000111 0100
D8.0	08	000 01000	111001 0100	000110 1011
D9.0	09	000 01001	100101 1011	100101 0100
D10.0	0A	000 01010	010101 1011	010101 0100
D11.0	0B	000 01011	110100 1011	110100 0100
D12.0	0C	000 01100	001101 1011	001101 0100
D13.0	0D	000 01101	101100 1011	101100 0100
D14.0	0E	000 01110	011100 1011	011100 0100
D15.0	0F	000 01111	010111 0100	101000 1011
D16.0	10	000 10000	011011 0100	100100 1011
D17.0	11	000 10001	100011 1011	100011 0100
D18.0	12	000 10010	010011 1011	010011 0100
D19.0	13	000 10011	110010 1011	110010 0100
D20.0	14	000 10100	001011 1011	001011 0100
D21.0	15	000 10101	101010 1011	101010 0100
D22.0	16	000 10110	011010 1011	011010 0100
D23.0	17	000 10111	111010 0100	000101 1011
D24.0	18	000 11000	110011 0100	001100 1011
D25.0	19	000 11001	100110 1011	100110 0100
D26.0	1A	000 11010	010110 1011	010110 0100
D27.0	1B	000 11011	110110 0100	001001 1011
D28.0	1C	000 11100	001110 1011	001110 0100
D29.0	1D	000 11101	101110 0100	010001 1011
D30.0	1E	000 11110	011110 0100	100001 1011
D31.0	1F	000 11111	101011 0100	010100 1011
D0.1	20	001 00000	100111 1001	011000 1001
D1.1	21	001 00001	011101 1001	100010 1001
D2.1	22	001 00010	101101 1001	010010 1001
D3.1	23	001 00011	110001 1001	110001 1001
D4.1	24	001 00100	110101 1001	001010 1001
D5.1	25	001 00101	101001 1001	101001 1001
D6.1	26	001 00110	011001 1001	011001 1001
D7.1	27	001 00111	111000 1001	000111 1001
D8.1	28	001 01000	111001 1001	000110 1001
D9.1	29	001 01001	100101 1001	100101 1001

Table A.1(b) Valid data code-groups				
Code Group Name	**Octet Value**	**Octet Bits HGF EDCBA**	**Current RD** abcdei fghj	**Current RD +** abcdei fghj
D10.1	2A	001 01010	010101 1001	010101 1001
D11.1	2B	001 01011	110100 1001	110100 1001
D12.1	2C	001 01100	001101 1001	001101 1001
D13.1	2D	001 01101	101100 1001	101100 1001
D14.1	2E	001 01110	011100 1001	011100 1001
D15.1	2F	001 01111	010111 1001	101000 1001
D16.1	30	001 10000	011011 1001	100100 1001
D17.1	31	001 10001	100011 1001	100011 1001
D18.1	32	001 10010	010011 1001	010011 1001
D19.1	33	001 10011	110010 1001	110010 1001
D20.1	34	001 10100	001011 1001	001011 1001
D21.1	35	001 10101	101010 1001	101010 1001
D22.1	36	001 10110	011010 1001	011010 1001
D23.1	37	001 10111	111010 1001	000101 1001
D24.1	38	001 11000	110011 1001	001100 1001
D25.1	39	001 11001	100110 1001	100110 1001
D26.1	3A	001 11010	010110 1001	010110 1001
D27.1	3B	001 11011	110110 1001	001001 1001
D28.1	3C	001 11100	001110 1001	001110 1001
D29.1	3D	001 11101	101110 1001	010001 1001
D30.1	3E	001 11110	011110 1001	100001 1001
D31.1	3F	001 11111	101011 1001	010100 1001
D0.2	40	010 00000	100111 0101	011000 0101
D1.2	41	010 00001	011101 0101	100010 0101
D2.2	42	010 00010	101101 0101	010010 0101
D3.2	43	010 00011	110001 0101	110001 0101
D4.2	44	010 00100	110101 0101	001010 0101
D5.2	45	010 00101	101001 0101	101001 0101
D6.2	46	010 00110	011001 0101	011001 0101
D7.2	47	010 00111	111000 0101	000111 0101
D8.2	48	010 01000	111001 0101	000110 0101
D9.2	49	010 01001	100101 0101	100101 0101
D10.2	4A	010 01010	010101 0101	010101 0101
D11.2	4B	010 01011	110100 0101	110100 0101
D12.2	4C	010 01100	001101 0101	001101 0101
D13.2	4D	010 01101	101100 0101	101100 0101
D14.2	4E	010 01110	011100 0101	011100 0101
D15.2	4F	010 01111	010111 0101	101000 0101
D16.2	50	010 10000	011011 0101	100100 0101
D17.2	51	010 10001	100011 0101	100011 0101
D18.2	52	010 10010	010011 0101	010011 0101
D19.2	53	010 10011	110010 0101	110010 0101
D20.2	54	010 10100	001011 0101	001011 0101
D21.2	55	010 10101	101010 0101	101010 0101
D22.2	56	010 10110	011010 0101	011010 0101
D23.2	57	010 10111	111010 0101	000101 0101

Table A.1(c) Valid data code-groups				
Code Group Name	**Octet Value**	**Octet Bits HGF EDCBA**	**Current RD** abcdei fghj	**Current RD +** abcdei fghj
D24.2	58	010 11000	110011 0101	001100 0101
D25.2	59	010 11001	100110 0101	100110 0101
D26.2	5A	010 11010	010110 0101	010110 0101
D27.2	5B	010 11011	110110 0101	001001 0101
D28.2	5C	010 11100	001110 0101	001110 0101
D29.2	5D	010 11101	101110 0101	010001 0101
D30.2	5E	010 11110	011110 0101	100001 0101
D31.2	5F	010 11111	101011 0101	010100 0101
D0.3	60	011 00000	100111 0011	011000 1100
D1.3	61	011 00001	011101 0011	100010 1100
D2.3	62	011 00010	101101 0011	010010 1100
D3.3	63	011 00011	110001 1100	110001 0011
D4.3	64	011 00100	110101 0011	001010 1100
D5.3	65	011 00101	101001 1100	101001 0011
D6.3	66	011 00110	011001 1100	011001 0011
D7.3	67	011 00111	111000 1100	000111 0011
D8.3	68	011 01000	111001 0011	000110 1100
D9.3	69	011 01001	100101 1100	100101 0011
D10.3	6A	011 01010	010101 1100	010101 0011
D11.3	6B	011 01011	110100 1100	110100 0011
D12.3	6C	011 01100	001101 1100	001101 0011
D13.3	6D	011 01101	101100 1100	101100 0011
D14.3	6E	011 01110	011100 1100	011100 0011
D15.3	6F	011 01111	010111 0011	101000 1100
D16.3	70	011 10000	011011 0011	100100 1100
D17.3	71	011 10001	100011 1100	100011 0011
D18.3	72	011 10010	010011 1100	010011 0011
D19.3	73	011 10011	110010 1100	110010 0011
D20.3	74	011 10100	001011 1100	001011 0011
D21.3	75	011 10101	101010 1100	101010 0011
D22.3	76	011 10110	011010 1100	011010 0011
D23.3	77	011 10111	111010 0011	000101 1100
D24.3	78	011 11000	110011 0011	001100 1100
D25.3	79	011 11001	100110 1100	100110 0011
D26.3	7A	011 11010	010110 1100	010110 0011
D27.3	7B	011 11011	110110 0011	001001 1100
D28.3	7C	011 11100	001110 1100	001110 0011
D29.3	7D	011 11101	101110 0011	010001 1100
D30.3	7E	011 11110	011110 0011	100001 1100
D31.3	7F	011 11111	101011 0011	010100 1100
D0.4	80	100 00000	100111 0010	011000 1101
D1.4	81	100 00001	011101 0010	100010 1101
D2.4	82	100 00010	101101 0010	010010 1101
D3.4	83	100 00011	110001 1101	110001 0010
D4.4	84	100 00100	110101 0010	001010 1101
D5.4	85	100 00101	101001 1101	101001 0010

Table A.1(c) Valid data code-groups				
Code Group Name	**Octet Value**	**Octet Bits HGF EDCBA**	**Current RD**	**Current RD +**
			abcdei fghj	abcdei fghj
D6.4	86	100 00110	011001 1101	011001 0010
D7.4	87	100 00111	111000 1101	000111 0010
D8.4	88	100 01000	111001 0010	000110 1101
D9.4	89	100 01001	100101 1101	100101 0010
D10.4	8A	100 01010	010101 1101	010101 0010
D11.4	8B	100 01011	110100 1101	110100 0010
D12.4	8C	100 01100	001101 1101	001101 0010
D13.4	8D	100 01101	101100 1101	101100 0010
D14.4	8E	100 01110	011100 1101	011100 0010
D15.4	8F	100 01111	010111 0010	101000 1101
D16.4	90	100 10000	011011 0010	100100 1101
D17.4	91	100 10001	100011 1101	100011 0010
D18.4	92	100 10010	010011 1101	010011 0010
D19.4	93	100 10011	110010 1101	110010 0010
D20.4	94	100 10100	001011 1101	001011 0010
D21.4	95	100 10101	101010 1101	101010 0010
D22.4	96	100 10110	011010 1101	011010 0010
D23.4	97	100 10111	111010 0010	000101 1101
D24.4	98	100 11000	110011 0010	001100 1101
D25.4	99	100 11001	100110 1101	100110 0010
D26.4	9A	100 11010	010110 1101	010110 0010
D27.4	9B	100 11011	110110 0010	001001 1101
D28.4	9C	100 11100	001110 1101	001110 0010
D29.4	9D	100 11101	101110 0010	010001 1101
D30.4	9E	100 11110	011110 0010	100001 1101
D31.4	9F	100 11111	101011 0010	010100 1101
D0.5	A0	101 00000	100111 1010	011000 1010
D1.5	A1	101 00001	011101 1010	100010 1010
D2.5	A2	101 00010	101101 1010	010010 1010
D3.5	A3	101 00011	110001 1010	110001 1010
D4.5	A4	101 00100	110101 1010	001010 1010
D5.5	A5	101 00101	101001 1010	101001 1010
D6.5	A6	101 00110	011001 1010	011001 1010
D7.5	A7	101 00111	111000 1010	000111 1010
D8.5	A8	101 01000	111001 1010	000110 1010
D9.5	A9	101 01001	100101 1010	100101 1010
D10.5	AA	101 01010	010101 1010	010101 1010
D11.5	AB	101 01011	110100 1010	110100 1010
D12.5	AC	101 01100	001101 1010	001101 1010
D13.5	AD	101 01101	101100 1010	101100 1010
D14.5	AE	101 01110	011100 1010	011100 1010
D15.5	AF	101 01111	010111 1010	101000 1010
D16.5	B0	101 10000	011011 1010	100100 1010
D17.5	B1	101 10001	100011 1010	100011 1010
D18.5	B2	101 10010	010011 1010	010011 1010
D19.5	B3	101 10011	110010 1010	110010 1010

Table A.1(d) Valid data code-groups				
Code Group Name	**Octet Value**	**Octet Bits HGF EDCBA**	**Current RD** abcdei fghj	**Current RD +** abcdei fghj
D20.5	B4	101 10100	001011 1010	001011 1010
D21.5	B5	101 10101	101010 1010	101010 1010
D22.5	B6	101 10110	011010 1010	011010 1010
D23.5	B7	101 10111	111010 1010	000101 1010
D24.5	B8	101 11000	110011 1010	001100 1010
D25.5	B9	101 11001	100110 1010	100110 1010
D26.5	BA	101 11010	010110 1010	010110 1010
D27.5	BB	101 11011	110110 1010	001001 1010
D28.5	BC	101 11100	001110 1010	001110 1010
D29.5	BD	101 11101	101110 1010	010001 1010
D30.5	BE	101 11110	011110 1010	100001 1010
D31.5	BF	101 11111	101011 1010	010100 1010
D0.6	C0	110 00000	100111 0110	011000 0110
D1.6	C1	110 00001	011101 0110	100010 0110
D2.6	C2	110 00010	101101 0110	010010 0110
D3.6	C3	110 00011	110001 0110	110001 0110
D4.6	C4	110 00100	110101 0110	001010 0110
D5.6	C5	110 00101	101001 0110	101001 0110
D6.6	C6	110 00110	011001 0110	011001 0110
D7.6	C7	110 00111	111000 0110	000111 0110
D8.6	C8	110 01000	111001 0110	000110 0110
D9.6	C9	110 01001	100101 0110	100101 0110
D10.6	CA	110 01010	010101 0110	010101 0110
D11.6	CB	110 01011	110100 0110	110100 0110
D12.6	CC	110 01100	001101 0110	001101 0110
D13.6	CD	110 01101	101100 0110	101100 0110
D14.6	CE	110 01110	011100 0110	011100 0110
D15.6	CF	110 01111	010111 0110	101000 0110
D16.6	D0	110 10000	011011 0110	100100 0110
D17.6	D1	110 10001	100011 0110	100011 0110
D18.6	D2	110 10010	010011 0110	010011 0110
D19.6	D3	110 10011	110010 0110	110010 0110
D20.6	D4	110 10100	001011 0110	001011 0110
D21.6	D5	110 10101	101010 0110	101010 0110
D22.6	D6	110 10110	011010 0110	011010 0110
D23.6	D7	110 10111	111010 0110	000101 0110
D24.6	D8	110 11000	110011 0110	001100 0110
D25.6	D9	110 11001	100110 0110	100110 0110
D26.6	DA	110 11010	010110 0110	010110 0110
D27.6	DB	110 11011	110110 0110	001001 0110
D28.6	DC	110 11100	001110 0110	001110 0110
D29.6	DD	110 11101	101110 0110	010001 0110
D30.6	DE	110 11110	011110 0110	100001 0110
D31.6	DF	110 11111	101011 0110	010100 0110
D0.7	E0	111 00000	100111 0001	011000 1110
D1.7	E1	111 00001	011101 0001	100010 1110

Table A.1(e) Valid data code-groups				
Code Group Name	**Octet Value**	**Octet Bits HGF EDCBA**	**Current RD**	**Current RD +**
			abcdei fghj	abcdei fghj
D2.7	E2	111 00010	101101 0001	010010 1110
D3.7	E3	111 00011	110001 1110	110001 0001
D4.7	E4	111 00100	110101 0001	001010 1110
D5.7	E5	111 00101	101001 1110	101001 0001
D6.7	E6	111 00110	011001 1110	011001 0001
D7.7	E7	111 00111	111000 1110	000111 0001
D8.7	E8	111 01000	111001 0001	000110 1110
D9.7	E9	111 01001	100101 1110	100101 0001
D10.7	EA	111 01010	010101 1110	010101 0001
D11.7	EB	111 01011	110100 1110	110100 1000
D12.7	EC	111 01100	001101 1110	001101 0001
D13.7	ED	111 01101	101100 1110	101100 1000
D14.7	EE	111 01110	011100 1110	011100 1000
D15.7	EF	111 01111	010111 0001	101000 1110
D16.7	F0	111 10000	011011 0001	100100 1110
D17.7	F1	111 10001	100011 0111	100011 0001
D18.7	F2	111 10010	010011 0111	010011 0001
D19.7	F3	111 10011	110010 1110	110010 0001
D20.7	F4	111 10100	001011 0111	001011 0001
D21.7	F5	111 10101	101010 1110	101010 0001
D22.7	F6	111 10110	011010 1110	011010 0001
D23.7	F7	111 10111	111010 0001	000101 1110
D24.7	F8	111 11000	110011 0001	001100 1110
D25.7	F9	111 11001	100110 1110	100110 0001
D26.7	FA	111 11010	010110 1110	010110 0001
D27.7	FB	111 11011	110110 0001	001001 1110
D28.7	FC	111 11100	001110 1110	001110 0001
D29.7	FD	111 11101	101110 0001	010001 1110
D30.7	FE	111 11110	011110 0001	100001 1110
D31.7	FF	111 11111	101011 0001	010100 1110

APPENDIX B

FINITE STATE MACHINES

The finite state machine for the PCS modules is presented here, notice that those are direct copy from the IEEE 802.3z standard for the 1000BASE-X.

The following State machines are presented:

- PCS transmit ordered-set state diagram.
- PCS transmit code-group state diagram.
- PCS receive state diagram, part a.
- PCS receive state diagram, part b.
- Carrier sense state diagram.
- Synchronization state diagram.
- Auto negotiation state diagram.

For details about the variables defined and the definition of state diagram usage ,refer to [6],

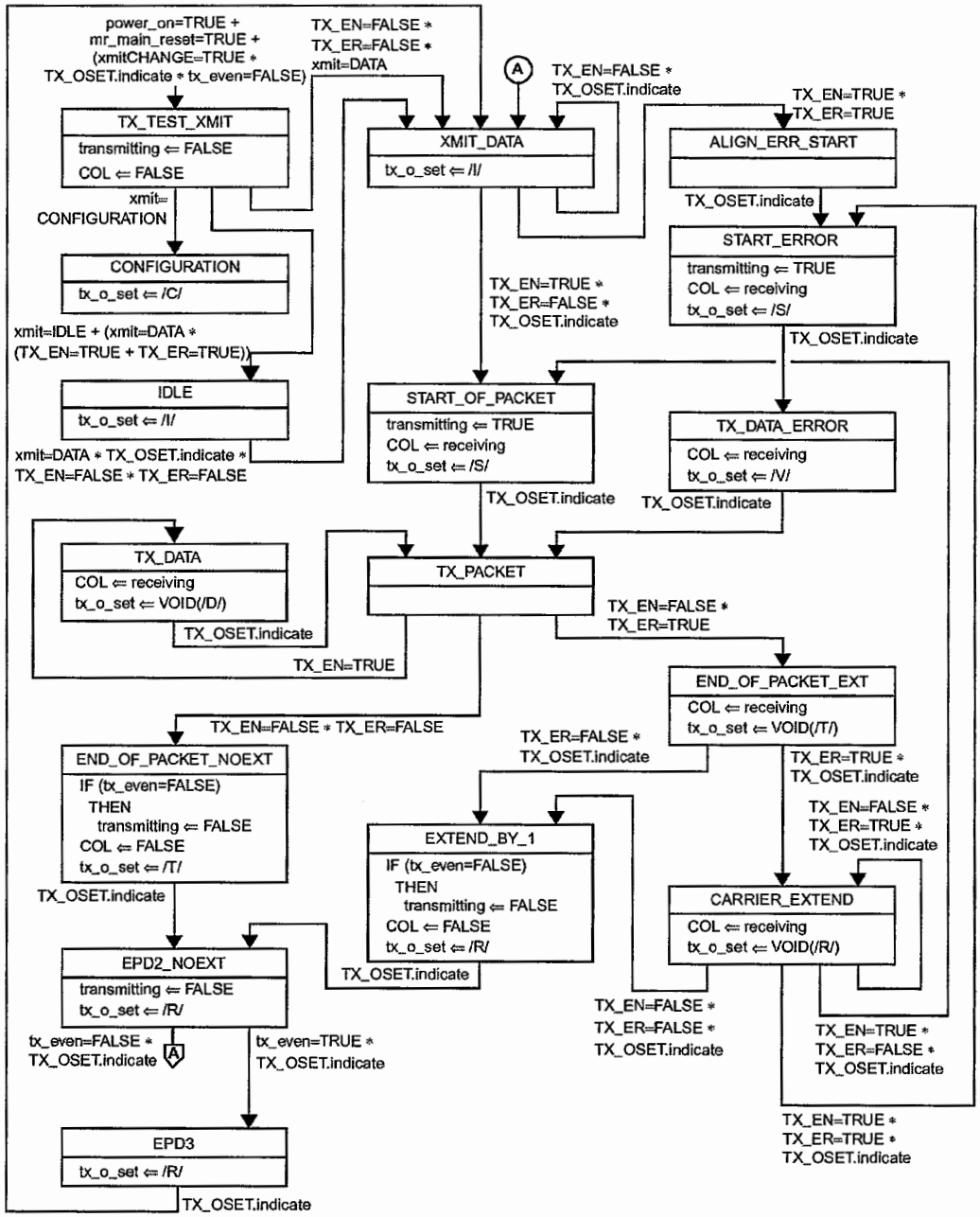


Figure B.1 PCS transmit ordered_set State diagram

(From IEEE Std. 802.3z 1999 All rights reserved [6])

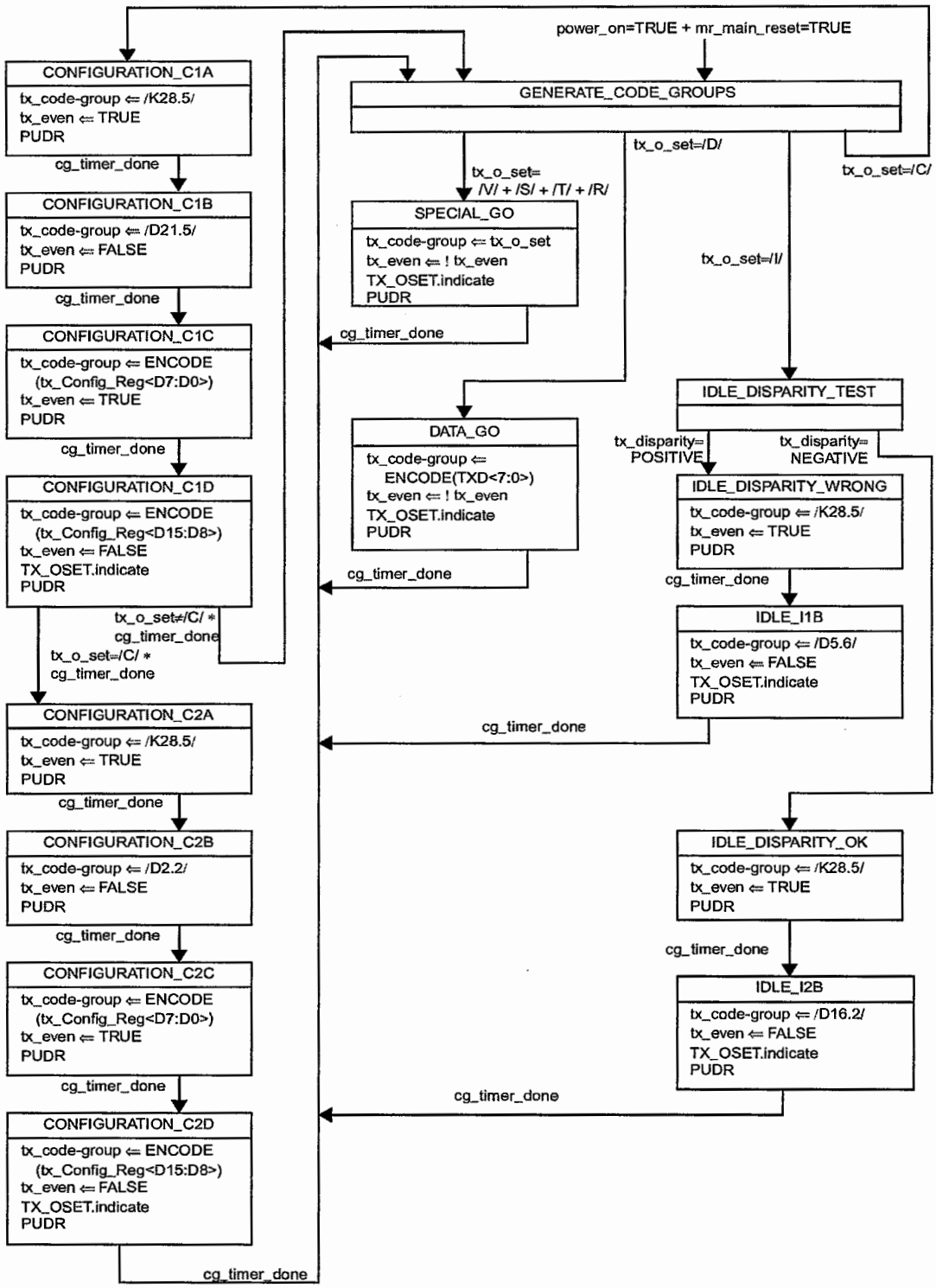


Figure B.2 PCS transmit code-group State diagram

(From IEEE Std. 802.3z 1999 All rights reserved [6])

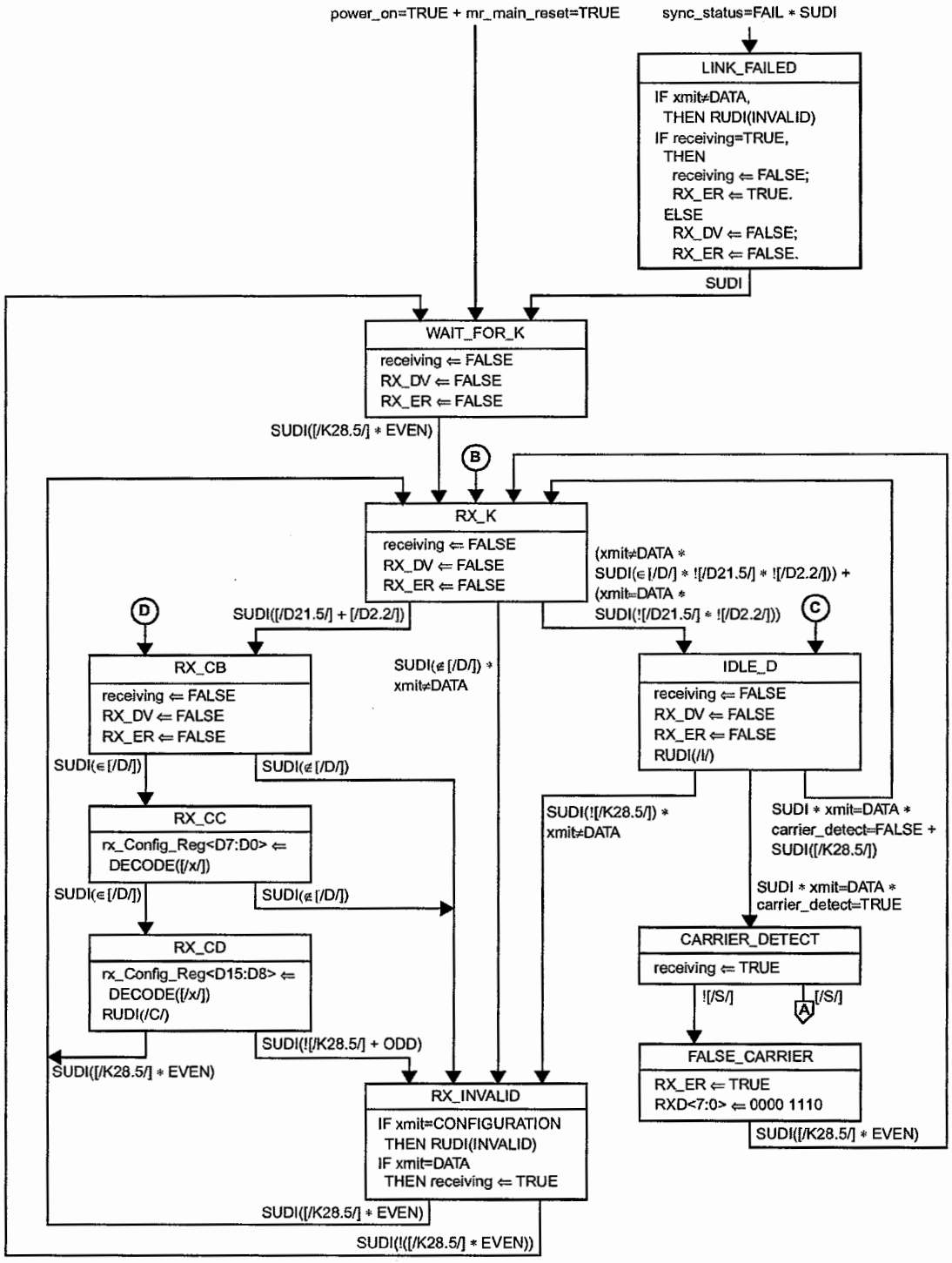


Figure B.3 Receive State diagram, part a

(From IEEE Std. 802.3z 1999 All rights reserved [6])

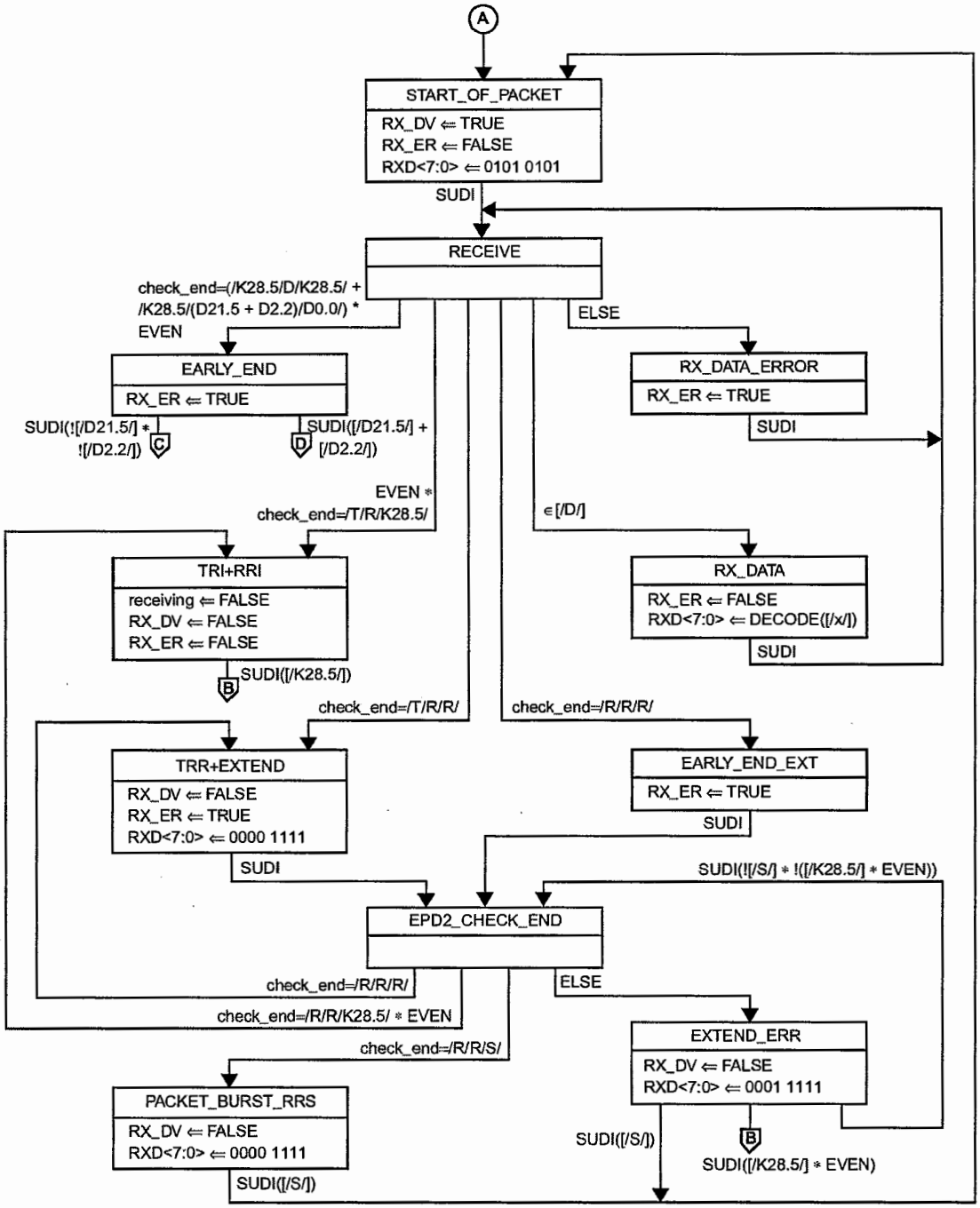


Figure B.4 Receive State diagram, part b

(From IEEE Std. 802.3z 1999 All rights reserved [6])

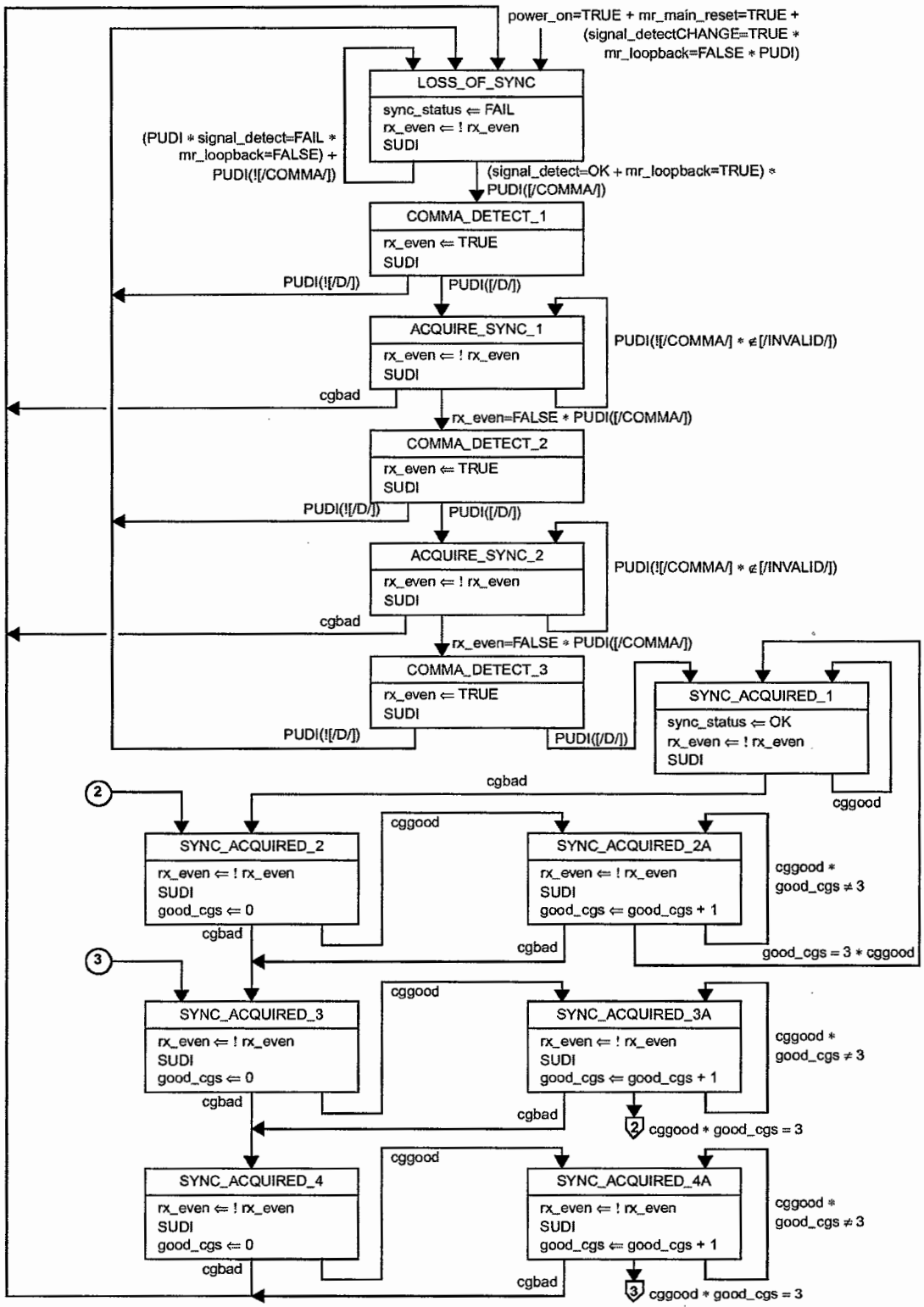


Figure B.5 Synchronization State diagram.

(From IEEE Std. 802.3z 1999 All rights reserved [6])

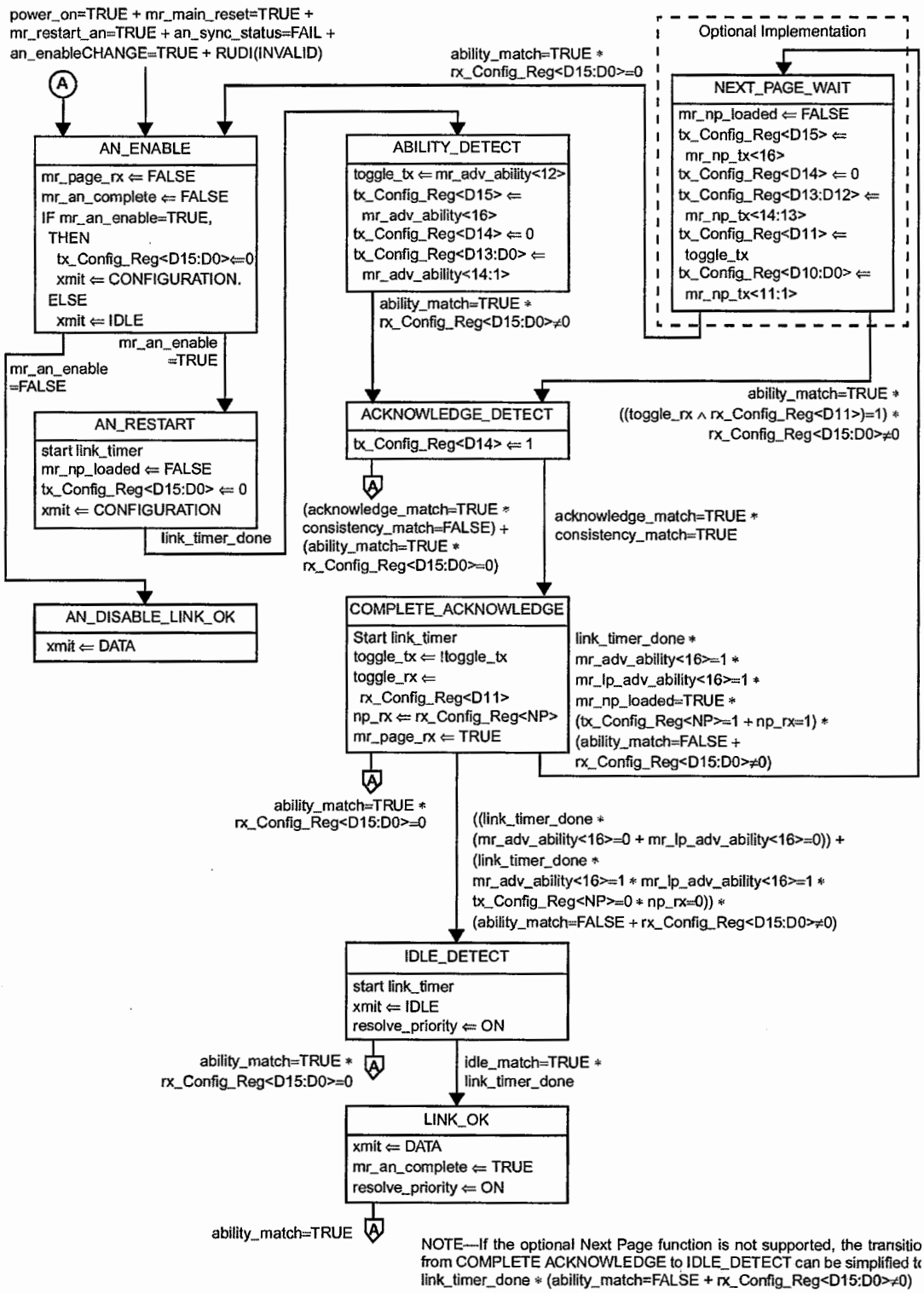


Figure B.6 Auto-Negotiation State diagram.

(From IEEE Std. 802.3z 1999 All rights reserved [6])

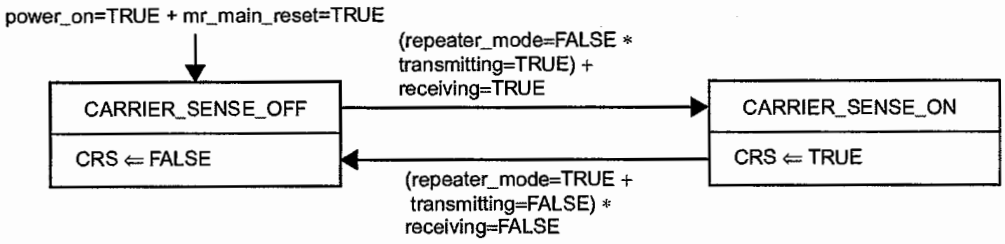


Figure B.7 Carrier sense State diagram.

(From IEEE Std. 802.3z 1999 All rights reserved [6])

APPENDIX C

SYNOPSYS SCRIPT FILES

Here we show the script files for the optimization of the PCS, note that all the modules were written into one file "all.v".

We have the following files

- compile.scr: in this file all the process of optimization is included.
- default.scr: this file has the clocks, inputs and outputs constraint and definitions.
- dont_use.scr: contain the cells that are not to be synthesized from.
- analyze.scr: contain the information about reading in the modules.
- regroup.scr: has the regroup command.

compile.scr

```
remove_design -all

include ./scripts/analyze.scr

include ./scripts/regroup.scr

include ./scripts/default.scr

report_design > logs/report_design.log

write -f db -hier -output ./DB/pcs_top.db pcs_top

compile

remove_unconnected_ports -blast_buses find(-hier cell "*")

fix_hold {GTX_CLK, PMA_RX_CLK, MDC}

compile -map_effort high -incremental -boundary_optimization

create_schematic -hierarchy -size infinite

write -f edif -hier -output ./DB/pcs_top.edif pcs_top

write -f verilog -hier -output ./pcs_top.v pcs_top

write_script -hier > pcs_top_compile.scr

write_timing -f sdf -context verilog -output ./DB/pcs_top.sdf

write -f db -hier -output ./DB/pcs_top_1stcmp.db pcs_top

check_test -verbose
check_design > logs/check_design_pc.log
report_design > logs/report_design_pc.log
report_constraints -all_violators -verbose > logs/report_violators.log
report_area > logs/area.log
report_reference > logs/reference.log
report_cell > logs/cell.log
report_net > logs/net.log
report_resources > logs/resources.log
report_port -verbose > logs/port.log
report_clock > logs/clock.log
report_timing -path full -delay max -max_paths 1 -nworst 1 > logs/worst_path.log
check_timing > logs/check_timings.log
```

analyze.scr

```
analyze -format verilog -lib WORK {"/home/bataineh/projects/pcs_top/syn/all.v"}

elaborate pcs_top > logs/elaborate.log
check_design > logs/check_design.log

uniquify

/* -output ./databases/pcs_top.db */
write -hier -l DATABASES pcs_top
```

regroup.scr

```
current_design = pcs_top

ungroup -all -flatten

check_design > logs/check_design_regroup.log

write -hier -l DATABASES pcs_top
```

dont_use.scr

```
/* fs80a_a/ND2 */

set_dont_use
find(cell,{fs80a_a/JK*,fs80a_a/NDL2*,fs80a_a/NRL2*,fs80a_a/DB*,fs80a_a/QDB
*,fs80a_a/QDL*,fs80a_a/QJK*,fs80a_a/QT*,fs80a_a/T*})

/* set_dont_use
find(cell,{fs80a_a/OAI112*,fs80a_a/OAI13*,fs80a_a/OAI22*,fs80a_a/OAI23*,fs80
a_a/OAI33*}) */

/* set_dont_use find(cell, {fs80a_a/ND2*}) for some reason with this in, incorrect
logic is generated */
```

default.scr page(1)

```
include dont_use.scr

derive_clocks

create_clock -name "GTX_CLK" -period 8 -waveform { "0" "4" } {   "GTX_CLK" }
set_clock_skew -rise_delay 1 "GTX_CLK"
set_clock_skew -fall_delay 1 "GTX_CLK"
set_dont_touch_network find( clock, "GTX_CLK")
set_clock_skew -uncertainty 1.0 "GTX_CLK"

create_clock -name "PMA_RX_CLK" -period 8 -waveform { "0" "4" } { "PMA_RX_CLK"}
set_clock_skew -rise_delay 1 "PMA_RX_CLK"
set_clock_skew -fall_delay 1 "PMA_RX_CLK"
set_dont_touch_network find( clock, "PMA_RX_CLK")
set_clock_skew -uncertainty 1.0 "PMA_RX_CLK"

create_clock -name "MDC" -period 400 -waveform {   "0" "200" } {   "MDC" }
set_clock_skew -fall_delay 1 "MDC"
set_clock_skew -rise_delay 1 "MDC"
set_dont_touch_network find( clock, "MDC")
set_clock_skew -uncertainty 1.0 "MDC"

set_dont_touch {PMA_RX_CLK,GTX_CLK,MDC}

set_false_path -setup -reset_path -to { "MDC" } -from { "GTX_CLK" }
set_false_path -hold -reset_path -to { "MDC" } -from { "GTX_CLK" }
set_false_path -setup -reset_path -to { "GTX_CLK" } -from { "MDC" }
set_false_path -hold -reset_path -to { "GTX_CLK" } -from { "MDC" }

/*set_false_path -setup -reset_path -to { "GTX_CLK" } -from { "PMA_RX_CLK" }*/
/*set_false_path -hold -reset_path -to { "GTX_CLK" } -from { "PMA_RX_CLK" }*/
/*set_false_path -setup -reset_path -to { "PMA_RX_CLK" } -from { "GTX_CLK" }*/
/*set_false_path -hold -reset_path -to { "PMA_RX_CLK" } -from { "GTX_CLK" }*/

set_false_path -from find(port,"phy_add")
set_false_path -from find(port,"power_on_in")

set_false_path -from find(port,"repeater_mode")
set_false_path -from find(port,"signal_detect")
```

default.scr page(2)

```
set_input_delay -clock GTX_CLK  -max -rise 2.5 { TX_ER, TX_EN, TXD}
set_input_delay -clock GTX_CLK  -max -fall 2.5 { TX_ER, TX_EN, TXD}
set_input_delay -clock GTX_CLK  -min -rise 1.0 { TX_ER, TX_EN, TXD}
set_input_delay -clock GTX_CLK  -min -fall 1.0 { TX_ER, TX_EN, TXD}

set_input_delay -clock PMA_RX_CLK -max -rise 2.5 rx_code_group
set_input_delay -clock PMA_RX_CLK -max -fall 2.5 rx_code_group
set_input_delay -clock PMA_RX_CLK -min -rise 1.0 rx_code_group
set_input_delay -clock PMA_RX_CLK -min -fall 1.0 rx_code_group

set_false_path -to find(port,"RX_CLK")
set_false_path -to find(port,"mr_loopback")

set_output_delay -clock GTX_CLK  -max -rise 2.5 {COL, CRS,tx_code_group}
set_output_delay -clock GTX_CLK  -max -fall 2.5 {COL, CRS,tx_code_group}
set_output_delay -clock GTX_CLK  -min -rise 0.5 {COL, CRS,tx_code_group}
set_output_delay -clock GTX_CLK  -min -fall 0.5 {COL, CRS,tx_code_group}

set_output_delay -clock PMA_RX_CLK  -max -rise 2.0 { RXD, RX_DV, RX_ER}
set_output_delay -clock PMA_RX_CLK  -max -fall 2.0 { RXD, RX_DV, RX_ER}
set_output_delay -clock PMA_RX_CLK  -min -rise 0.5 { RXD, RX_DV, RX_ER}
set_output_delay -clock PMA_RX_CLK  -min -fall 0.5 { RXD, RX_DV, RX_ER}


set_input_delay -clock MDC -max 10 "MDIO"
set_input_delay -clock MDC -min 10 "MDIO"

set_output_delay -clock MDC -max 10 "MDIO"
set_output_delay -clock MDC -min 10 "MDIO"


set_drive  1 all_inputs()
set_max_capacitance  5 all_outputs()

set_max_area 0

set_max_transition 1.2 current_design

set_operating_conditions -min BCCOM -max WCCOM

set_fix_multiple_port_nets -all -buffer_constants
```

REFERENCES

1. A. Doboli, A. Nunez-Aldana, N. Dhanwada, S. Gansen, and R. Vemuri, "Behavioral Synthesis of Analog Systems using Two-Layered Design Exploration", Proc. Of the 36th Design Automation Conference, pp. 951-957.
2. P. Campisi, "A CMOS Analog Cell Library for Analog Synthesis Systems", Master of Science Thesis, University of Cincinnati, 1998.
3. J. Kadambi, I. Crayford, and M. Kalkunte, "Gigabit Ethernet: Migrating to High-Bandwidth LANs", Prentice-Hall, Upper Saddle River, NJ, 1998.
4. R. Seifert, "Gigabit Ethernet: technology and applications for high speed LANs", Addison-Wesley, Reading, MA, 1998.
5. S. Saunders, "Data Communications Gigabit Ethernet Handbook", McGraw-Hill, New York, NY 1998.
6. IEEE Std 802.3z-1998, "Local and Metropolitan Area Networks", Institute of Electrical and Electronics Engineers, Inc., New York, NY, 1998.
7. J. M. Lee, "Verilog® Quickstart", Kluwer Academic Publishers, Norwell, MA, 1997
8. S. Palnitkar, "Verilog® HDL: A Guide to Digital Design and Synthesis", SunSoft Press a Prentice Hall Title, Mountain View, California, 1996.
9. "SpectreHDL® Reference Manual", Version 4.3.4 June 1995., Cadence Design Systems, Inc., San Jose, CA 1995.
10. Analogy Inc. Oct 99, http://www.analogy.com/pubs/guide/6.html

11. "Analog Artist® Mixed-Signal Design Environment User Guide", product Version 4.4.3 December 1998, Cadence Design Systems, Inc., San Jose, CA 1998.

12. P. Kurup, T. Abbasi, "Logic Synthesis Using Synopsys®", 2nd edition, Kluwer Academic Publisher, Norwell, MA, 1997.

13. D. Knapp, "Behavioral Synthesis: Digital System Design Using The Synopsys® Behavioral Compiler™" , Prentice Hall, Upper Saddle River, NJ, 1996.

14. IEEE Std. 802.3u-1995, "Local and Metropolitan Area Networks", Institute of Electrical and Electronics Engineers, Inc., New York, NY, 1995.

15. A. X.Widmar, P. A. Franszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code", IBM. Res. Develop., Vol. 27, No. 5,pp 440-451, September 1983.

16. Synopsys® "CHIP Synthesis", Synopsys, Inc. Mountain View CA, 1996.

17. Huimin Xia, Khaldoun Bataineh, Marwan Hassoun and Joe Kryzak, "A Mixed –signal Behavioral Level Implementation of 1000BASE-X Physical Layer for Gigabit Ethernet",1999 IEEE International Symposium on Circuits and Systems (ISCAS'99) (Orlando,FL), volume I, pp I-431 – I-434, May/June 1999.